

AD-A037 834

HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
ADVANCED LOGIC TECHNOLOGY.(U)

UNCLASSIFIED

FEB 77 G A ANDERSON, E D JENSEN, R Y KAIN

F30602-75-C-0148

F0375-FR

RADC-TR-76-388

NL

1 OF 4
ADA037834



ADA037834

RADC-TR-76-388
Final Technical Report
February 1977

ADVANCED LOGIC TECHNOLOGY

Honeywell Inc



Approved for public release; distribution unlimited.



ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

DDC FILE COPY

ADVANCED LOGIC TECHNOLOGY

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED:

Oskar A. Reimann

OSKAR A. REIMANN
Project Engineer

APPROVED:

Alan R. Barnum

ALAN R. BARNUM
Asst Chief, Information Sciences Division

FOR THE COMMANDER:

John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

ACCESSION BY	DATE	BY	REMARKS
NTIS			
DOC			
UNANNOUNCED			
JUSTIFICATION			
DISTRIBUTION/AVAILABILITY CODES		AVAIL. and/or SPECIAL	
DISC.		A	

Do not return this copy. Retain or destroy.

This report has been reviewed and approved for publication.

Chas. A. Kennaum

APPROVED:

Alan R. Bauman

FOR THE COMMANDER:

John P. Huss

RECEIVED FOR
 HTS
 DDC
 UNANNOUNCED
 JUSTIFICATION

White Section
 Buff Section

BY ☒ ☐ ☐

DISTRIBUTION / AVAILABILITY CODES

DIST. ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐

AVAILABLE OR SPECIAL

A

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-76-388	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ADVANCED LOGIC TECHNOLOGY,	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report, 15 Mar 75 - 14 Aug 76	6. PERFORMING ORG. REPORT NUMBER F0375-FR
7. AUTHOR(s) G. A./Anderson, K. W./Krause E. D./Jensen, J. A./White R. Y./Kain, L. B. Wing	8. CONTRACT OR GRANT NUMBER(s) F30602-75-C-0148	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Honeywell Inc/Systems & Research Center, 2600 Ridgway Parkway N. E. Minneapolis MN 55413	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 45941216	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCA) Griffiss AFB NY 13441	12. REPORT DATE February 1977	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same	13. NUMBER OF PAGES 346	
	15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		
18. SUPPLEMENTARY NOTES RADC Project Engineer: Oskar A. Reimann (ISCA)		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Content Addressed Memory Query Languages Associative Memory Interpreter Large Data Bases Computer Architecture		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes an extended-capability content-addressed memory (CAM) which operates on formatted data bases up to around 10^9 bits in size. The system, called ECAM, consists of a Control Unit and a serial associative store with logic blocks in the 100-gate complexity range provided at each word. The organization is relatively independent of storage technology; the current design assumes charge-coupled device (CCD) memory organized in 4K-bit words, shifted at a 1-microsecond rate. The Control Unit is microprogrammed to interpret a		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

402 349

1B

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

block-structure query language which operates on logical data structures, such as Codd's relations. The mapping of logical structures to operations on physical storage is performed by the hardware. A full repertoire of associative search and arithmetic operators is provided. In its target application, the ECAM will require less than one hour per day to perform a task that was estimated to require between 40 and 700 hours per day on a large commercial mainframe.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

	<u>Page</u>
SECTION 1 INTRODUCTION AND SUMMARY	1
1.1 Background	1
1.2 Hardware Structure	3
1.2.1 Control Unit	3
1.2.2 Array	5
1.3 Application and System Software	8
1.4 Application Analysis and Performance Summary	11
1.4.1 Query Workload	11
1.4.2 Update Workload	12
1.4.3 Host Operating System Communication Overhead	13
1.4.4 Workload Summary	13
1.5 Recommended Program Plan	13
1.6 Report Organization	15
SECTION 2 SACWARDANS SYSTEM REQUIREMENTS	16
2.1 SACWARDANS Processing Model	16
2.1.1 Directory Types	17
2.1.2 Processing Operations	19
2.2 SACWARDANS Directories Data Base	20
2.2.1 Directory Content	20
2.2.2 Directory Sizes	25
2.3 Directory Algorithms	25
2.3.1 Maintenance of the Directories Data Base	26
2.3.2 Trend and Correlation Analysis	44
2.3.3 Workload	51
2.4 Requirements Summary	63
2.4.1 Data Base Storage Requirements	63
2.4.2 Algorithms	65
2.4.3 Use Factors	80
SECTION 3 SOFTWARE STRUCTURE	81
3.1 Host Functions	81
3.2 Master Minicomputer Functions	83
3.3 Slave Processor	84
SECTION 4 ECAM HARDWARE STRUCTURE	85
4.1 Control Unit	87
4.1.1 Master Control Processor	87
4.1.2 Slave Controller	89
4.1.3 Host Interface	127

TABLE OF CONTENTS (CONCLUDED)

	<u>Page</u>
4.2 ECAM Associative Array	128
4.2.1 Array Physical Overview	129
4.2.2 Word Logic LSI Chip	140
4.2.3 Hybrid Circuit Module	171
4.2.4 Array Storage Board	174
4.2.5 Control Generator Logic Tree	177
4.2.6 Associative Array Summary	180
 SECTION 5 SYSTEM OPERATION	 183
5.1 SACWARDANS Directory Storage	183
5.1.1 SACWARDANS Directories Organization in ECAM Array	183
5.1.2 Data Base Size	188
5.1.3 Checkpoint, Restart, and Recovery	192
5.2 Slave Controller Functions	197
5.2.1 Programmer Visible Data Structures	199
5.2.2 Operations Available to the Programmer	201
5.2.3 Implementation Data Structures	216
5.2.4 Instruction Interpretation	222
5.2.5 Firmware Register Assignments	244
 SECTION 6 HARDWARE IMPLEMENTATION	 250
6.1 Memory Unit Cabinet	250
6.1.1 Power Supplies and Cooling	255
6.1.2 Storage Board	255
6.1.3 Storage Buffer Board	268
6.1.4 Cabinet Buffer Board	268
6.1.5 Test Interface Board	268
6.2 Control Unit Cabinet	271
6.3 Peripheral Cabinet	278
6.4 Test Methodology and Equipment	278
 SECTION 7 PROGRAM PLAN	 287
7.1 Introduction	287
7.2 Schedule	287
7.2.1 Phase I Schedule	287
7.2.2 Phase II Schedule	289
7.3 Detail Hardware Development Task Plan	289
7.4 System Software Subcontractor SOW	338
7.4.1 Scope	338
7.4.2 Statement of Work	338
7.4.3 Furnished Equipment and Facilities	341

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1	ECAM Structure	4
2	Proposed Packaging Scheme	5
3	ECAM Associative Word	6
4	Memory Board Packaging Scheme	7
5	Stored Data Structure	8
6	Logical to Physical Mapping	9
7	Program Plan Summary	14
8	Add Location	28
9	Modify Location	30
10	Delete Location	31
11	Add Organization	32
12	Modify Organization	33
13	Delete Organization	34
14	Add Equipment	35
15	Modify Equipment	37
16	Delete Equipment	38
17	Add Location/Organization/Equipment	39
18	Modify Location/Organization/Equipment	40
19	Delete Location/Organization/Equipment	41
20	Add Event	42
21	Add Message	45
22	Purge Messages and Events	46
23	First Sample Query	48
24	Second Sample Query	49
25	Third Sample Query	50
26	Simple Search Segment	52
27	More-Complex Search Segment	52
28	Count Responders Segment	53
29	Count Responders Exceed Threshold Segment	53

LIST OF ILLUSTRATIONS (CONTINUED)

<u>Figure</u>		<u>Page</u>
30	Analog Count Responders Segment	54
31	Analog Count Responders Exceed Threshold Segment	54
32	Sum Field Segment	55
33	Sum Field Exceed Threshold Segment	55
34	Return All Responders Segment	56
35	Return First N Responders Segment	56
36	Return Next N Responders Segment	56
37	Typical Query Search	57
38	Typical Query with More Complexity	58
39	Typical Query with Threshold Check	59
40	Typical Query with Return of Events of Threshold Exceeded	60
41	Procedure for Preparing a Chart of Activity Over Time	61
42	Average Activity in the Indicated Period	62
43	ECAM System Software Major Functions	82
44	Control Unit Block Diagram	88
45	Interpreter Structure	90
46	Fetch Unit Registers Primarily Related to Driving Register Busses	94
47	Fetch Unit Memories and Associated Registers	95
48	Other Fetch Unit Registers	96
49	Microinstruction Format O -- Operate	98
50	Type 1 Instruction Format	106
51	Parameter and Instruction Buffering	118
52	Iteration Control Sequences	119
53	IT Controller Microcode Formats	120
54	Iteration Controller Register Structure	122
55	IT Controller Control Logic	123
56	Storage Controller Functions	126
57	Memory and Word Logic Blocks	128

LIST OF ILLUSTRATIONS (CONTINUED)

<u>Figure</u>		<u>Page</u>
58	I/O Switch Data Connections	130
59	I/O Switch Control Connections	131
60	ECAM Array Signal Distribution	132
61	ECAM Storage Board Function Block Diagram	134
62	ECAM Hybrid Circuit Module Function Block Diagram	135
63	ECAM Word Logic LSI Chip Function Block Diagram	137
64	ECAM Word Logic Block	137
65	ECAM Match Memory	138
66	ECAM I/O Switch	139
67	I/O and Multiple Match Resolver Control Logic	140
68	Word Logic LSI Chip Function Block Diagram	141
69	ECAM Word Logic Block	142
70	Match State Information Transfers	146
71	ECAM Word Logic Match Memory	159
72	ECAM Word Logic I/O Switch	159
73	I/O Switch Operation Concept	160
74	Selector Matrix	161
75	Shifter Matrix	162
76	I/O Switch Mode Control	163
77	Word Logic Interface Block	164
78	Selector/Shifter Blocks B1 and B3	166
79	Selector/Shifter Block B2	166
80	Physical Data Line Interface Block	167
81	I/O and MMR Control Logic	168
82	Decoder	169
83	Encoder	169
84	Word Logic Match Count Adder	170
85	Word Logic Function Code Decoding Logic	171
86	Hybrid Circuit Module	172
87	Array Storage Board Function Block Diagram	175

LIST OF ILLUSTRATIONS (CONTINUED)

<u>Figure</u>		<u>Page</u>
88	Storage Board Control Generator Logic Tree	176
89	Hybrid Interface Block	181
90	Control Generator Logic Block	182
91	ECAM Word Formatting Approach	183
92	Format 1	185
93	Format 2	185
94	Format 3	185
95	Format 4	186
96	Format 5	186
97	Formats 1 through 5 Consolidated	187
98	First Sample Format Field	189
99	Second Sample Format Field	190
100	Third Sample Format Field	191
101	Checkpoint Flowchart	195
102	Logical Data Structures	200
103	Activation Block Format	208
104	Record Descriptor Structures	217
105	Field Descriptor	217
106	Field Descriptor Table	218
107	Match Stack	219
108	Addressing Within Activation Blocks	220
109	Addressing Data Structures	221
110	Alternative Field Organization and Shift Direction Possibilities	226
111	An Assignment of Registers into Two Files	247
112	Typical ECAM System	251
113	Floor Plan Concept	252
114	Memory Unit Cabinet Interconnect Scheme	253
115	Memory Unit Cabinet	254
116	Memory Unit Card Assignments	256

LIST OF ILLUSTRATIONS (CONTINUED)

<u>Figure</u>		<u>Page</u>
117	Memory Unit Cabinet Interior Layout	257
118	3-inch x 3-inch Hybrid With T-10 Chip as Storage Device	258
119	Memory Board Assembly	260
120	Memory Unit Storage Board	261
121	Memory Unit Hybrid Assembly	262
122	CD-460 Hybrid Layout	263
123	Hybrid Packaging Concept	265
124	Word Logic LSIC	266
125	ECAM Memory Unit Storage Buffer Board	269
126	Memory Unit Cabinet Buffer Board	270
127	Control Unit Cabinet	272
128	Control Unit Cabinet Interior Layout	273
129	Control Unit Card Assignments	274
130	Control Unit Cabinet Signal Distribution Board	276
131	Master Signal Distribution Board	277
132	Software Subcontractor's Software Development Facility for ECAM	279
133	Aerospace Division SDF/Test Facility for ECAM	280
134	Deliverable ECAM System Functional Block Diagram	281
135	Peripheral Cabinet	282
136	ECAM Phase I Master Schedule	288
137	ECAM Phase II Master Schedule	290
138	ECAM Program Overall Task Breakdown	291
139	Task 1.0 Breakdown -- Program Management	292
140	Task 2.0 Breakdown -- System Design	293
141	Task 3.0 Breakdown -- Electrical Design	294
142	Task 4.0 Breakdown -- Mechanical Design	295
143	Task 5.0 Breakdown -- System Software and Analysis	296
144	Task 6.0 Breakdown -- Support Software Development	297

LIST OF ILLUSTRATIONS (CONCLUDED)

<u>Figure</u>		<u>Page</u>
145	Task 7.0 Breakdown -- Procure/Fab	298
146	Task 8.0 Breakdown -- Test	299
147	Task 9.0 Breakdown -- Delivery and Acceptance	300
148	Task 10.0 Breakdown -- Field Support	300
149	Task 11.0 Breakdown -- Test Equipment	301

LIST OF TABLES

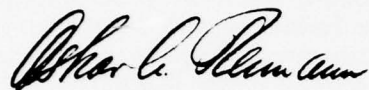
<u>Table</u>		<u>Page</u>
1	Word Logic Function Summary	7
2	Query Traffic	11
3	Equipment Index	21
4	Organization Index	22
5	Location Index	23
6	Message Index	23
7	Event Correlation Index	24
8	L/O/E Correlation Index	25
9	SACWARDANS Data Base Size	26
10	Distribution of Record Lengths	64
11	Distribution of Item Lengths	64
12	Use of Array in Algorithm for Add Equipment	65
13	Use of Array in Algorithm for Add Organization	66
14	Use of Array in Algorithm for Add Location	66
15	Use of Array in Algorithm for Add Location/ Organization/Equipment	67
16	Use of Array in Algorithm for Add Message	67
17	Use of Array in Algorithm for Add Event	68
18	Use of Array in Algorithm for Modify Equipment	69
19	Use of Array in Algorithm for Modify Organization	69
20	Use of Array in Algorithm for Modify Location	70
21	Use of Array in Algorithm for Modify Location/ Organization/Equipment	70
22	Use of Array in Algorithm for Delete Equipment	71
23	Use of Array in Algorithm for Delete Organization	71
24	Use of Array in Algorithm for Delete Location	71
25	Use of Array in Algorithm for Delete Location/ Organization/Equipment	72
26	Use of Array in Algorithm for Purge Messages and Events	72

LIST OF TABLES (CONTINUED)

<u>Table</u>		<u>Page</u>
27	Estimated Use of Array Operations for Data Base Management	73
28	Use of Array in Algorithm for First Sample Query	75
29	Use of Array in Algorithm for Second Sample Query	76
30	Use of Array in Algorithm for Third Sample Query	76
31	Query Workload Assuming Sample Queries	77
32	Query Workload Assuming Extreme of 11 Fields Searched per Query	78
33	Query Workload -- High Expected Value	
34	Query Workload -- High Expected Value, Including ID Return	79
35	Kernel Machine Interface Lines	91
36	Assigned Operand Addresses	92
37	Operand Addressing Modes	98
38	Preassigned OPOP Codes	101
39	Condition Register Bits Assigned by the Fetch Unit	108
40	Register Module Specifications	111
41	ALU Functions	112
42	IT Interface Register Assignment	114
43	Register Address Assignments	116
44	OPOP Codes for the Slave Controller	116
45	Condition Register Bit Assignments for Slave Controller	116
46	Match Manipulation Functions	144
47	ECAM Associative Array Function Summary	157
48	Use of ECAM Bytes in Sample Formats	192
49	Iteration Controller Commands Summary	229
50	Register Assignments within Register File A	248
51	Register Assignments within Register File B	249
52	ECAM Word Logic LSIC Estimated Area	267
53	Storage Device Alternatives	267
54	ECAM Minicomputer and Peripheral Requirements	283

EVALUATION

The "Advanced Logic Technology" study was aimed at developing an architecture solution for a large data base application embodied in the indications and warning problem. This effort developed a detailed register-level definition of a content addressed memory to solve that problem. A program plan for implementing such a system was also developed. The study results will be an input for further trade-off and analysis prior to making a decision to implement a system to meet operational requirements.



OSCAR A. REIMANN
Project Engineer

SECTION 1

INTRODUCTION AND SUMMARY

This is the final report describing the results of a study conducted by Honeywell Inc., Systems and Research Center (S&RC), for the United States Air Force, Rome Air Development Center (RADC), under contract F30602-75-C-0148. The objective of the study was the architectural definition of an Extended Content-Addressed Memory (ECAM) suited for large data base applications where conventional software solutions are inadequate. The requirements for a proposed system called the SAC Warning Data Analysis System (SACWARDANS) were used as a baseline for machine development. Informatics, Inc. assisted Honeywell in the definition of these requirements.

The major results of the effort are: (1) the register-level definition of the ECAM, and (2) a plan for its development. These and other results are summarized in this section, and recommendations are presented. Details are presented in succeeding sections of this report.

1.1 BACKGROUND

Motivation for the development of the Extended Content-Addressed Memory came from the data base requirements of an application at the Strategic Air Command (SAC) called Indications and Warning Support. This system, called the SAC Warning Data Analysis System (SACWARDANS), was scheduled for an initial operating capability in 1978 and would allow intelligence analysts at display consoles to make on-line queries to a data base containing "messages" pertaining to various activities of interest. Approximately 3000 messages per day are added to the data base. Messages are retained in the system for a minimum of 30 days before being deleted. This results in an average data base on the order of 10^9 to 10^{10} bits. SACWARDANS is considered typical of the high-performance data base requirements of the U. S. intelligence community.

An analysis of the processing requirements for SACWARDANS was performed for RADC by Planning Research Corp. (PRC) and Informatics, Inc. under contract F30602-73-C-0359. This work, documented in PRC report WPO217 dated January 1974, showed that conventional serial processor approaches could not meet system performance requirements. In the report, three approaches were compared: a conventional Honeywell Information Systems HIS-6080 implementation, the Goodyear STARAN with a backing disc containing data base directories, and the ECAM, which, at that time, was in the conceptual design phase at Honeywell. The report's conclusion was:

"To summarize, the ECAM is expected to provide superior performance and in many ways result in a high-quality system which can serve both operational requirements and also provide a vehicle for developing new methodologies in the subject area of SACWARDANS."

The ECAM was shown to offer speed advantages of from 125:1 to 226:1 over the HIS-6080 processor used as a baseline, and storage efficiency improvements of 3.6:1. Compared with STARAN, the ECAM offers a speed advantage of approximately 3:1 and a storage efficiency improvement of 1.2:1.

The requirement for a device such as the ECAM stems from the inherent performance limitations in conventional data base approaches. Conventional data base systems are implemented on serial processors with limited amounts of fast memory. This has resulted in performance which deteriorates drastically as the data base size increases. Also, conventional memories are location-addressed, a fact which complicates the processing problem with issues not inherent in either the data or the system functional requirements. The major effect of location addressing has been increased storage overhead for index tables. In large data bases, management of these tables is a problem in its own right. Fast insertion and deletion of records requires that a minimum of tables be involved; fast retrieval of records requires that a large number of different attributes be indexed in the directories. Thus, the fast update requirement of SACWARDANS conflicts with the requirement for fast retrieval.

In contrast to conventional techniques, content-addressed memories like the ECAM have the capability of retrieving information directly, based on attributes of the data itself. This is done by including sufficient processing capability in the data storage medium to perform searching operations. The use of Content-Addressed Memories (CAMs) to overcome the constraints of conventional data base systems has been suggested by many, but, until recently, the cost of CAM systems has been prohibitively high. This high cost was due primarily to the cost of logic required for content addressability and the high cost of the storage itself. To date, 10^6 bits has been the upper limit on implemented CAMs, while SACWARDANS requirements call for capacity to approximately 10^9 bits.

In 1973, the contractor's ongoing work in large-scale integrated circuits (LSICs) indicated that advances in LSICs would allow systems of 10^9 bits to be built at reasonable cost by 1978, and that the historical limits on CAM sizes would no longer apply. Investigations were then begun into CAM organizations appropriate for such sizes. Thus, the overall ECAM concept was under internal development as a technology investigation prior to this contract effort. The effort expended under this contract has been directed toward refining the ECAM concepts into a register-level design expressly for data base applications, and towards developing the program plan.

1.2 HARDWARE STRUCTURE

The ECAM is a special-purpose machine designed to be attached to one or more host computers and to be used as an access processor for the data base it contains. The major functional units of the ECAM are shown in Figure 1. An artist's sketch of the proposed packaging is shown in Figure 2.

The machine is divided into two portions: the CAM array, and the control unit. The control unit is designed around a bus-organized minicomputer and includes the mini (called the master), a custom-designed controller for the array (called the slave), and one (or more) interfaces to the host(s). The array consists of a multiplicity of 4096-bit serial storage words, with combinational logic at each word to effect the associative functions.

1.2.1 Control Unit

The main unit within the control unit is the master minicomputer. Its memory bus provides the basic structure of the control unit, and the availability of standard software facilitates writing of application code to mediate between the host and the slave controller. Because of its ubiquity in intelligence applications, the contractor recommends that the PDP-11/45 be chosen as master control processor, but the control unit design is such that almost any bus-organized minicomputer could be used.

The ECAM/host interface is designed to connect to the host as a standard high-speed peripheral (such as a disc). It is controlled via the master's programmed input/output (I/O) facility, shown by the dotted line in Figure 1, and transfers blocks of information between the host and the master's memory in a transparent fashion. The design of this interface is obviously very dependent on the choice of a host; details have not been specified in this initial contract. In subsequent work, when a host has been selected and a final selection of master processor has been made, design of the interface can proceed in a straightforward manner.

The slave portion of the control unit is most critical to efficient operation of the array and, as such, has received the most attention during the control unit design effort. The two major subunits of the slave are the interpreter and the iteration control. The interpreter is a high-speed, microprogrammed unit which is designed specifically for interpretively executing a block-structured query language used to specify ECAM operation sequences. Query language sequences are passed from the master to the slave via buffers in the master's memory. The control store of the interpreter is writable, allowing easy changes to the query language. The output of the interpreter is a stream of array primitives which are passed to the iteration control unit via dedicated buffers within the slave. Iteration control is a hardwired subunit which generates control signal sequences to effect the array operations.

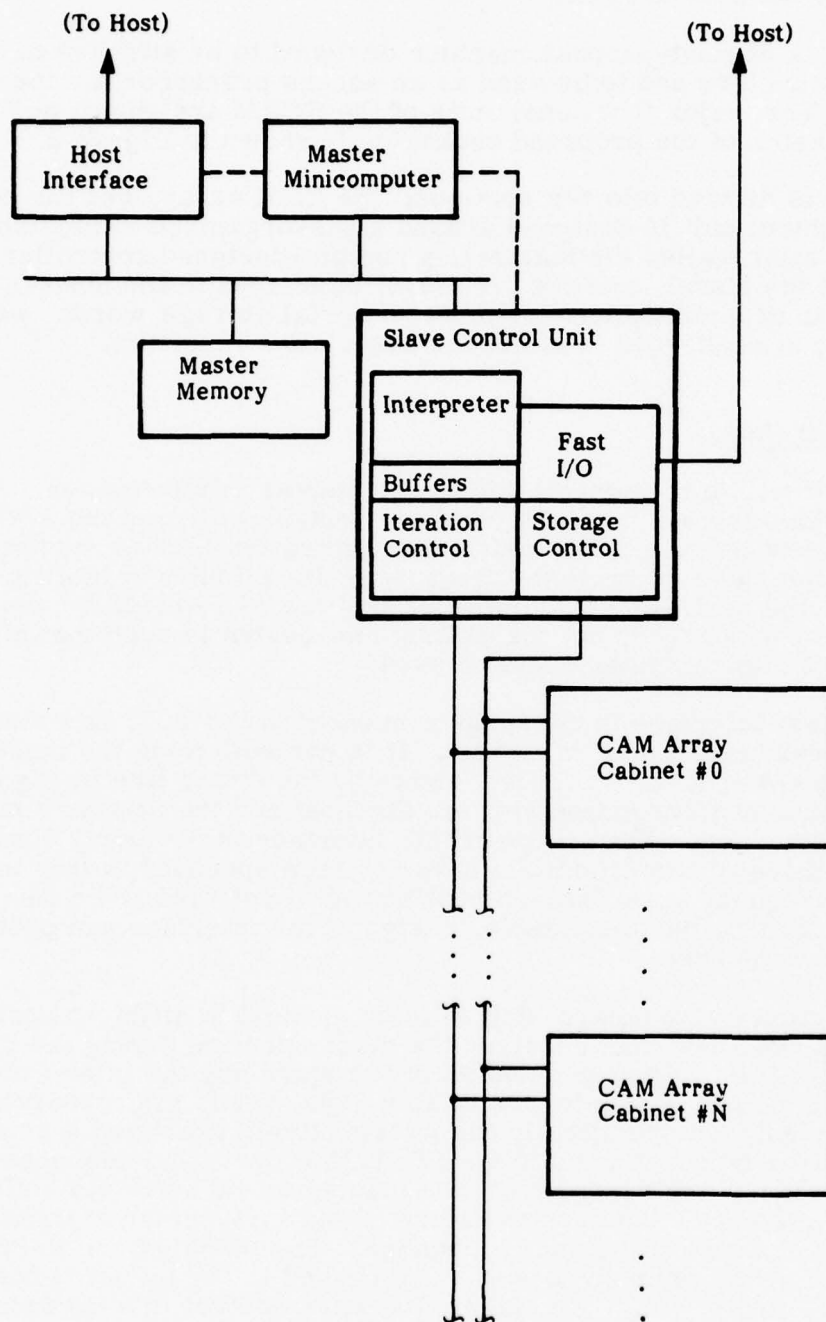


Figure 1. ECAM Structure

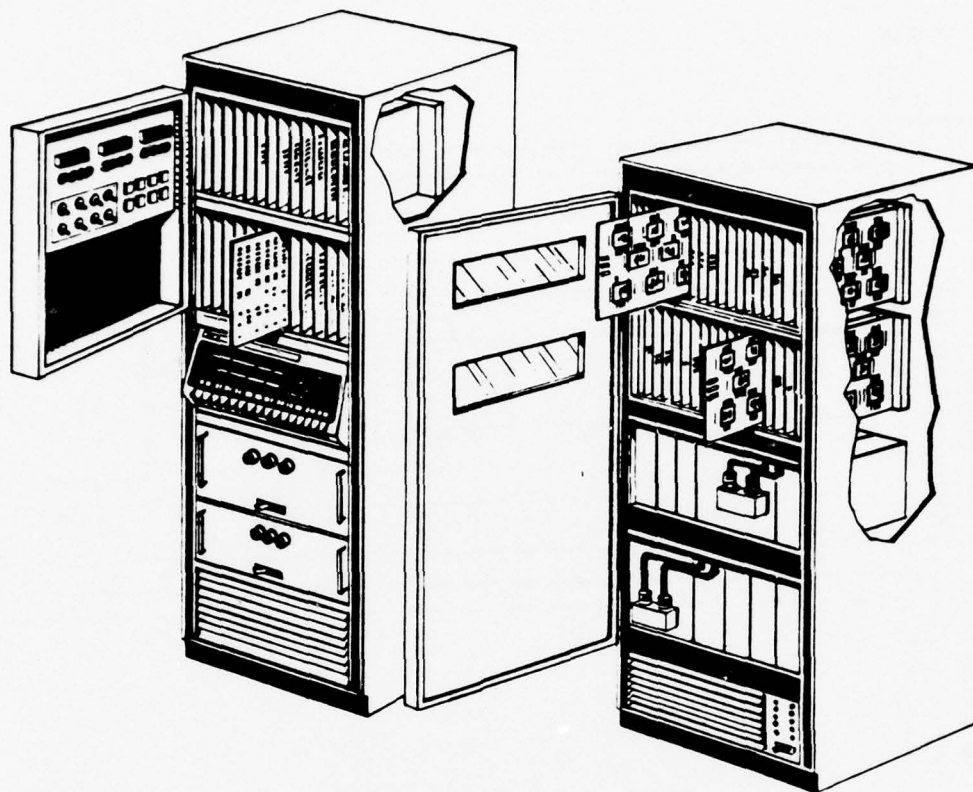


Figure 2. Proposed Packaging Scheme

To keep the bulk of the design independent of the storage technology, the storage control functions of addressing, shifting, refresh, and others, have been isolated to a single subunit of the slave. A high-bandwidth I/O capability is also provided, under control of a distinct subunit of the slave. As was the case for the main host interface, the fast I/O control design has been deferred to a subsequent effort because of its specificity to a particular host.

The complexity of the slave controller is estimated at 500 to 800 small and medium-scale integrated-circuit packages. It is designed for implementation in standard and Schottky transistor-transistor logic (TTL) circuit technology with clocks of 10 MHz.

1.2.2 Array

The array consists of a large number (up to 250,000) of associative words as shown in Figure 3. Each word consists of 4096 bits of charge-coupled device (CCD) storage, randomly addressable to 256-bit registers, and a block called

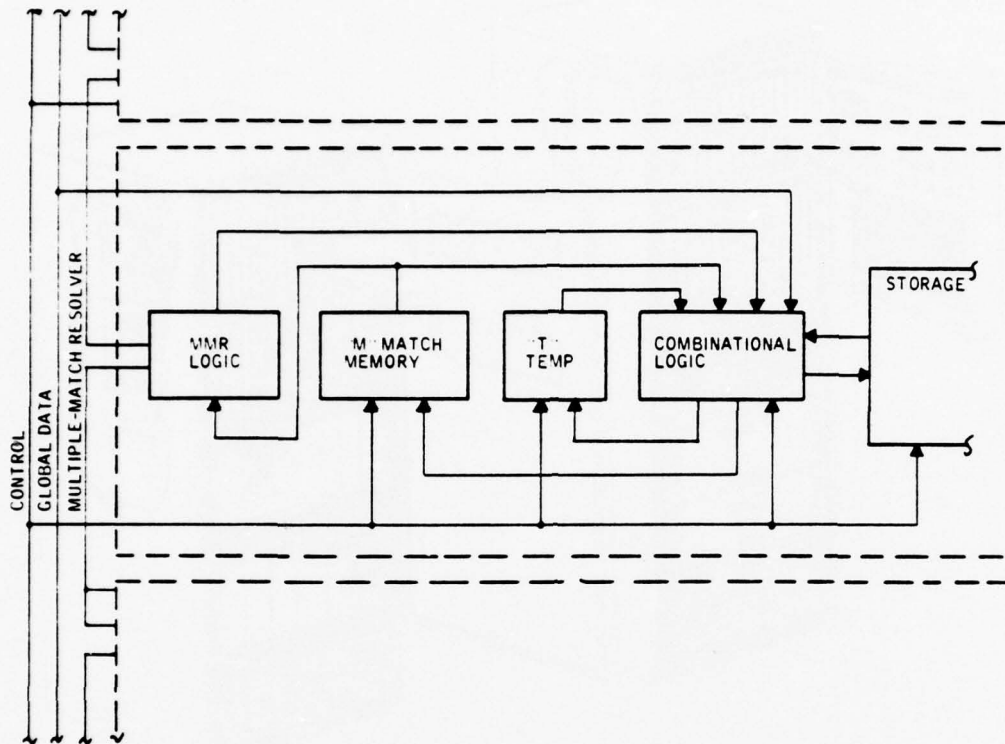


Figure 3. ECAM Associative Word

the "word logic" which supports the content addressing and associative functions of the array. The two major elements of the word logic are the match memory and the arithmetic-logic block. Word logic operations such as searches and arithmetic are performed by selecting one of 16 match bits from the memory and repeatedly executing a sequence of combinational operations on each bit of a field within the storage word. For most operations, the inputs to the combinational logic are the selected match bit, a "global" data signal from the control unit, and the "local" data bit from the storage part. A complete list of the word logic equations is presented and discussed in Section 4.2; a summary of functions is presented in Table 1.

The ECAM packaging baseline assumes that the storage is contained on LSICs of 10 words by 4096 bits. This is within the capability of present CCD technology of many vendors, including Honeywell. (In the baseline, we have assumed a shift rate of 1 microsecond per bit. This is somewhat slower than current technology capabilities, but was chosen to simplify signal distribution.) Each storage chip is paired with a word logic chip containing 10 word logic blocks together with first-level support logic for the multiple-match resolver, match counting, and I/O facilities. At the next level of

Table 1. Word Logic Function Summary

Operation	Word Logic Function
Processing	<ul style="list-style-type: none"> • Add/Subtract • Reverse Subtract • Arithmetic Compare • Minimum/Maximum
Input/Output	<ul style="list-style-type: none"> • Input • Output • Output and Tag Duplicates • Broadcast Input • Output ORing
State manipulation	<ul style="list-style-type: none"> • 17 logical functions between Match, T, and other word logic state variables
I/O and MMR control	<ul style="list-style-type: none"> • Five functions for I/O, MMR, and Match Counting

packaging, eight storage/word logic pairs are mounted on hybrid substrates. These substrates are then placed on conventional circuit cards. An artist's sketch of the packaging scheme is shown in Figure 4.

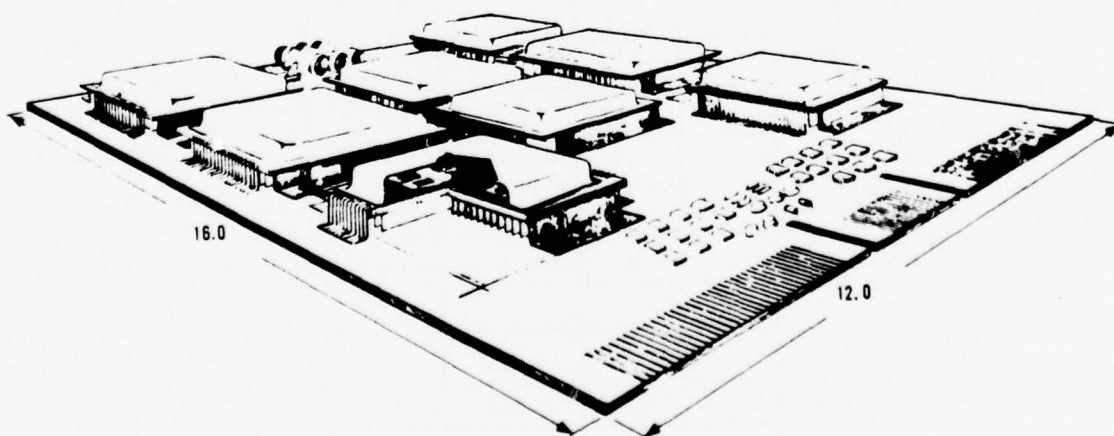


Figure 4. Memory Board Packaging Scheme

Although the baseline ECAM storage technology is CCD, the design deliberately has been kept as technology-independent as possible. Alternative approaches are discussed in Section 6. The basic constraint on the storage medium is a requirement for stop/start and read-modify-write capability on a per-bit basis. Beyond this, the speed, cost, power consumption, and mechanical attributes of the ECAM may be varied by changes in the storage technology. For instance, use of magnetic bubble storage would allow ECAM sizes to increase to perhaps 10^{10} to 10^{11} bits with a corresponding reduction in speed.

In addition to the word logic shown in Figure 3, the ECAM is provided with a high-speed I/O path which allows 10 words to be logically selected to participate simultaneously in a single input or output operation. The switch which implements this fast I/O mode is included at the word logic chip level. The effective transfer bandwidth of the ECAM is raised from 10^6 bits per second to 10^7 bits per second by use of the fast I/O mode.

1.3 APPLICATION AND SYSTEM SOFTWARE

The ECAM is primarily intended to operate on data stored as tables consisting of a number of fixed-size records, each subdivided into fields of varying length. The design is consistent with the relational view of data, where the tables are the relations, the records are the n-tuples, and the fields contain domain values. An example of this basic structure is shown in Figure 5. In an application such as SACWARDANS, the table sizes are as indicated in the figure. As discussed in Section 2, approximately 5 to 10 differently formatted directories will coexist within the ECAM. Both binary integers and characters are used as field values; within the ECAM, they are treated uniformly as bit strings.

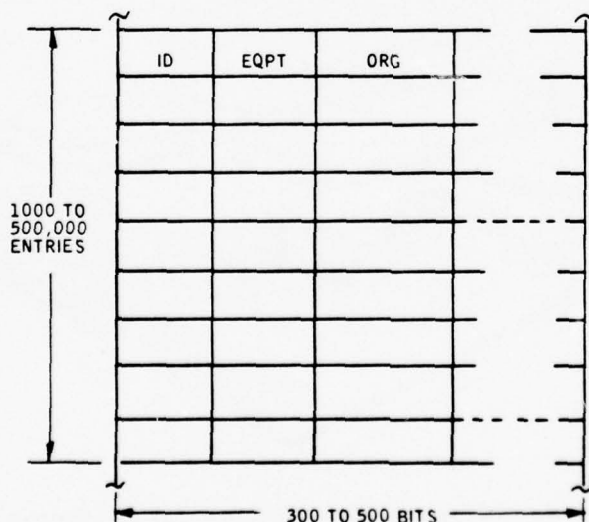


Figure 5. Stored Data Structure

The division of functions between host and ECAM is made as follows: terminal handlers and user job interfaces within the host support the generation of query sequences. Once such a sequence has been prepared, it is transferred, together with identification tags, to the ECAM via the host's standard I/O subsystem hardware and software. The query sequence references the logical structure of the data stored in the ECAM and may, in addition, refer to intermediate search results left by the user after a previous sequence.

The master control processor is responsible for management of pending sequences and for transmitting the results of queries back to the host. This includes allocation of master memory buffers for incoming sequences, scanning of sequences to create code for the interpreter, and allocation of master memory buffers for results being returned by the slave. In addition, users requiring temporary storage of results during the period between two queries may request temporary storage areas within the ECAM. The master control processor has responsibility for management of these areas. The scheduling and dispatching of code blocks to the interpreter is also the responsibility of the master. The slave is "multiprogrammed," in that code blocks may include "WAIT"-type operations which relinquish control, but no preemption is allowed.

The mapping of the logical table structures onto the physical storage is as shown in Figure 6. A small number of physical word formats are defined, each having a different combination of directory entries. A word may contain one or more of the directories. The particular packing strategy chosen will result from a tradeoff of speed and storage efficiency for a given set of directory widths and lengths. The choice of a packing strategy is a data base administration function; changes are expected to occur infrequently.

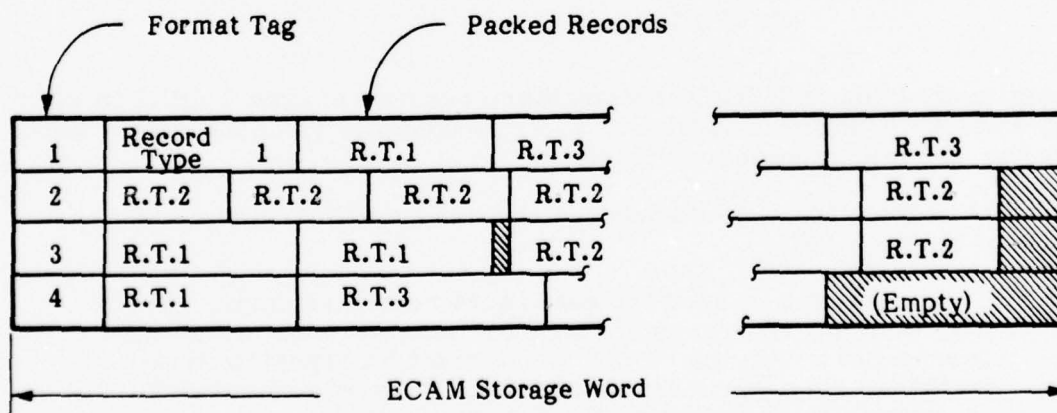


Figure 6. Logical to Physical Mapping

The ECAM word formats are defined by a set of descriptor tables stored in the master's memory. These consist of: (1) a Record Type table listing attributes of the various records (n-tuples) and pointing to (2) Record Instance Lists describing alternative physical placements of each record type. Finally, a Field Descriptor Table provides information on the placement of fields within records. These various tables are described in detail in Section 5.

The function of the interpreter is to execute the code sequences received from the master, including mapping references to the logical data structures into references to the physical storage scheme by the use of the descriptor tables. This involves creation of loops to sequence through multiple instances of records and modification of programmer-specified loops to optimize shifting of the array.

An example of the slave's language is shown in the example intermediate language sequence that follows. The program marks that record containing

```
FOR ALL RECORDS (TYPE1) DO
    FIND(VALUE1, FIELD1)
    PUSH1
    FIND(VALUE2, FIELD2)
    OR
    MAXIMUM(FIELD3)
ENDFOR
```

the maximum value in FIELD3 among those records of type TYPE1 in which either FIELD1 contains VALUE1 or FIELD2 contains VALUE2. This language has the following characteristics:

- It is strictly block-structured and can be interpreted using a single stack.
- The programmer sees the data in its relational form. There may be many instances of "TYPE1" records over which the operations within the "FOR" block must be repeated; these instances may occur in various word formats and various positions, as specified by the descriptor tables.
- The programmer sees the match memory (Figure 3) as a stack; between operations, these stacks are saved with the record instances.

The ECAM word formats are defined by a set of descriptor tables stored in the master's memory. These consist of: (1) a Record Type table listing attributes of the various records (n-tuples) and pointing to (2) Record Instance Lists describing alternative physical placements of each record type. Finally, a Field Descriptor Table provides information on the placement of fields within records. These various tables are described in detail in Section 5.

The function of the interpreter is to execute the code sequences received from the master, including mapping references to the logical data structures into references to the physical storage scheme by the use of the descriptor tables. This involves creation of loops to sequence through multiple instances of records and modification of programmer-specified loops to optimize shifting of the array.

An example of the slave's language is shown in the example intermediate language sequence that follows. The program marks that record containing

```
FOR ALL RECORDS (TYPE1) DO
    FIND(VALUE1, FIELD1)
    PUSH1
    FIND(VALUE2, FIELD2)
    OR
    MAXIMUM(FIELD3)
ENDFOR
```

the maximum value in FIELD3 among those records of type TYPE1 in which either FIELD1 contains VALUE1 or FIELD2 contains VALUE2. This language has the following characteristics:

- It is strictly block-structured and can be interpreted using a single stack.
- The programmer sees the data in its relational form. There may be many instances of "TYPE1" records over which the operations within the "FOR" block must be repeated; these instances may occur in various word formats and various positions, as specified by the descriptor tables.
- The programmer sees the match memory (Figure 3) as a stack; between operations, these stacks are saved with the record instances.

The syntax and semantics of the intermediate language are completely defined by the microcode stored in the slave. No part of the language interface has been hardwired. Rather, the hardware deliberately has been kept general, so that experience with the operational system can be used to make improvements in the master/slave language interface.

The software is structured and its functions divided among the system components in a hierarchical manner: the host need not know any details of the database structure or the query language implementation; the master need not know details of the iterations through bit positions to scan fields; the iteration controller handles these lowest-level details in a predetermined way. By this, we feel we have succeeded in moving complexities to the lowest possible level, while simultaneously designing to allow change in all critical areas.

1.4 APPLICATION ANALYSIS AND PERFORMANCE SUMMARY

The ECAM workload is largely comprised of two functions, Queries and Updates. The requirements and performance for each area are discussed below.

1.4.1 Query Workload

The SACWARDANS applications analysis in Section 2 of this document characterizes a "high" query level of 65,000 queries per day. We can use the average load estimated from this query level (shown in Table 2) to determine the average traffic at several crucial points within the ECAM system.

Table 2. Query Traffic

Operation	65,000 Queries/Day
Parameters to transfer to ECAM	~412,000
Bytes of parameters	~900,000
Searches	~350,000
Exact match searches	~260,000
Between limits searches	~ 25,000
Greater than searches	~ 65,000
Results to be returned to host	~110,000
Bytes to be returned to host	~325,000

Traffic between the host and master contains three components:

- Search specifications
- Parameter values
- Result values

Each search specification contains record selection information, search-type specifications, logical and control specifications, and result routing information. This information will require approximately 20 bytes per search. Thus, the approximately 350,000 searches per day become 7,000,000 bytes per day of search specifications.

Parameter values are 900,000 bytes per day, from the applications analysis. Result values are estimated at 325,000 bytes per day. Thus, the total master-host traffic is 8,225,000 bytes per day, or approximately 100 bytes per second.

Array shifting times are another load factor. These can be estimated by combining the parameter shifting times with estimates of the shift times required to position the memory for each operation.

The applications analysis produced some basic data needed to estimate the array shifting requirements. First, one must consider how many different instances of each type of record will exist within the formatted words. This is discussed in Section 2. Second, one must compute how many bit shifts will be required to sequence through the implied loops. Third, one must determine the number of record operations required per transaction (Section 2, Tables 12 through 16). Finally, one must find the expected query rates (Section 2.4.2).

By combining these numbers and reducing the averages to bit shifts per second, we find that the ECAM array will be shifting to find or search operands approximately 28,600 times per second. This is less than 3 percent of the maximum shift rate. The maximum rate that could be handled during crises situations is thus 35 times the average demand.

1.4.2 Update Workload

Using the applications analysis data in Section 2, Figure 31, and applying computations similar to those above, we determine the following worst-case numbers:

- Master-host traffic: 24 bytes per second
- Shifts: 13,408 shifts per second

Thus, the update workload is approximately 45 percent of the query workload. Note that some of the update workload can be deferred during crises.

1.4.3 Host Operating System Communication Overhead

At 50,000 transactions per day (estimate of the application's requirements), the GCOS operating system for an HIS-6080 would spend approximately 70 seconds of processor time per day constructing and handling messages to the ECAM system. This is approximately 10 times the amount of time required to transmit the messages between the two system components, but it is not a performance bottleneck.

1.4.4 Workload Summary

The total average workload requires 125 bytes per second communications between the master and host machines and 42,000 ECAM shifts per second. This represents approximately a 5-percent, or 1-hour-per-day, load on the ECAM. This result is similar to the one obtained by the original PRC/Informatics study team.

It is important to understand, however, that these figures assume a relatively idealized system. They should therefore be taken only as an indication that the ECAM is more than adequate for the baseline problem. More detailed timing information will be available from the simulations which begin the next-phase program.

1.5 RECOMMENDED PROGRAM PLAN

The contractor's recommendations for ECAM development are summarized in Figure 7. The plan is discussed in detail in Section 7 of this report. Basically, it consists of a two-phase effort covering a period of 3 years and resulting in an initial 10-million-byte system installed at an Air Force site.

The first phase of the program covers 1 year and has three major outputs. First, it results in quantitative data on system performance. Second, it carries the design work to a point where implementation costs for both software and hardware can be accurately estimated. Third, it includes hardware development work sufficient to minimize Phase 2 implementation risks. At the end of Phase 1, then, the government will have complete cost/benefit and risk information on the Phase 2 effort.

The Phase 1 effort covers three distinct areas: performance simulation, critical array component development, and storage element work. Of these, the performance simulation work may be separated as shown and performed prior to the others. The effect of this would be to delay the machine development, since the other two areas require 12-month efforts. Also, deferring the array and storage work would delay the availability of accurate cost and risk information.

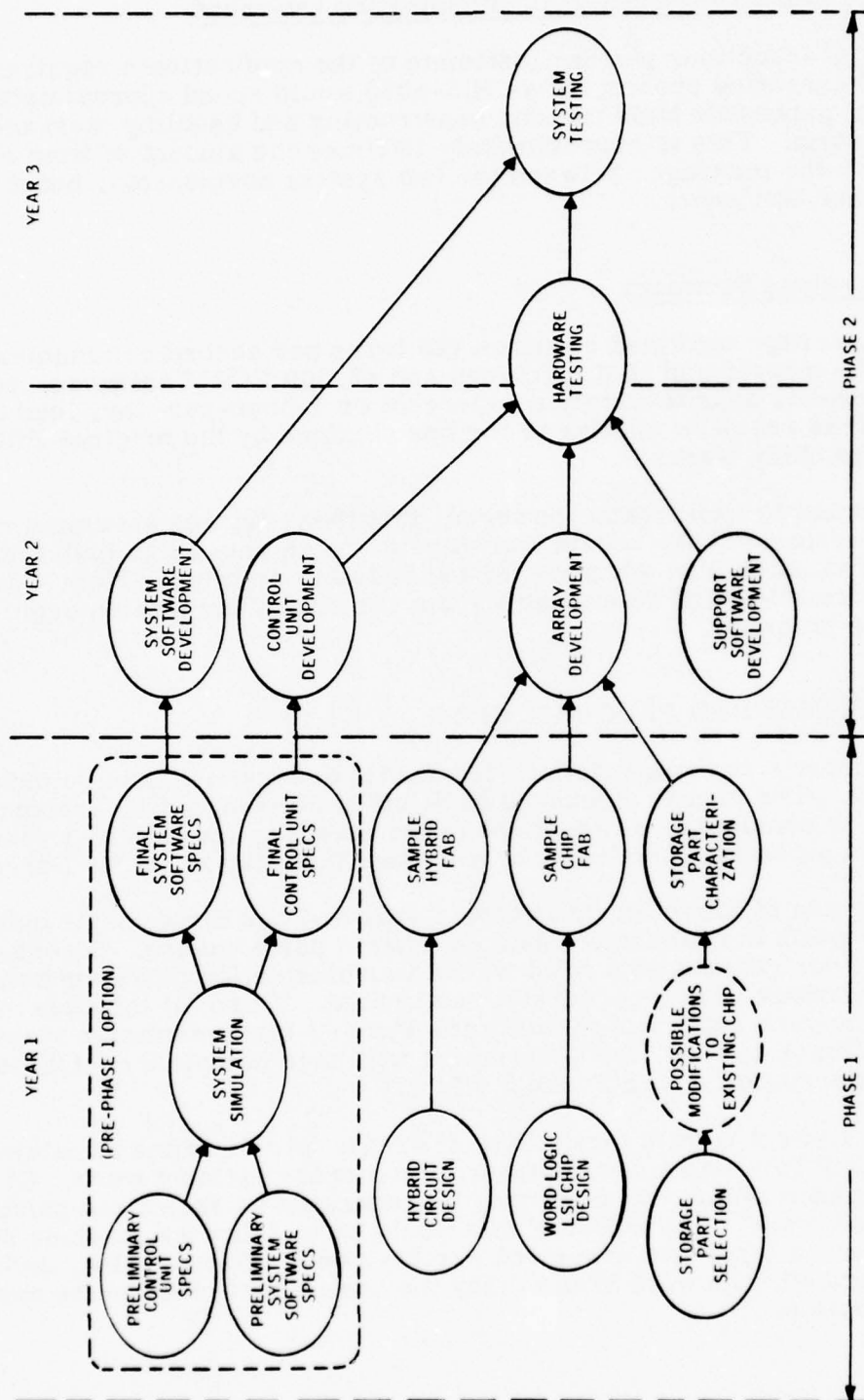


Figure 7. Program Plan Summary

The Phase 2 program, as shown in Figure 7, covers the development and testing of the initial 10-million-byte system. The baseline packaging approach allows for expansion of this basic system in 10-million-byte (cabinet) increments up to a maximum of 160 million bytes (1.28×10^9 bits).

1.6 REPORT ORGANIZATION

This report covers three general areas: the baseline requirements, the resulting system design, and the program plan.

Section 2 of the report was provided by Informatics, Inc. under subcontract. It describes the baseline SACWARDANS requirements and discusses how the processing can be mapped onto the ECAM system.

Section 3 introduces the software structure that is expected to be used with the ECAM.

Section 4 describes the register-level structure of the ECAM and includes many logic-level diagrams and flowcharts. Following this, Section 5 describes the details of the way the ECAM is controlled to operate as a data base machine.

Section 6 describes the mechanical baseline for the ECAM and includes a discussion of alternative storage technologies.

Finally, Section 7 presents a plan for ECAM development, including detailed task writeups and a software subcontractor Statement of Work.

SECTION 2

SACWARDANS SYSTEM REQUIREMENTS

Development of the ECAM design requires a functional specification of ECAM-related aspects of the SACWARDANS system. This section discusses the content and algorithms for manipulation of the SACWARDANS directory data base as it might appear and be used. The discussion herein is functional and it avoids discussion of either hardware or software design tradeoffs. Hence, the content of the directory data base is discussed, but not its mapping into associative storage. Mapping strategies are discussed in Section 5.1.

This section is divided into four subsections. Section 2.1 contains a paraphrase of the processing model previously developed and reported by Planning Research Corporation and Informatics Inc. The conceptual model is essentially the same as that developed before. The discussion in Section 2.1 expands upon the model description, adding foundation information not necessarily known to those unfamiliar with this type of application system. Section 2.2 discusses the SACWARDANS directory data base, showing its content and estimating non-overhead storage requirements. The content is described in a relational form as a system user would perceive it. Section 2.3 discusses the algorithms for maintaining and using the SACWARDANS directory data base. These are functionally specified, and specifics resulting from particular mappings of the data base into the associative storage are not presented. Section 2.4 discusses reliability, backup, and recovery requirements.

2.1 SACWARDANS PROCESSING MODEL

Previous SAC on-line intelligence systems have been developed to support cyclic processing on a multiday basis. In contrast, SACWARDANS is to support minute-to-minute analysis and reporting of worldwide events. It requires that historical data be correlated with incoming events to maintain, in near real-time, an accurate and comprehensive picture of what is going on in the world and how it affects all SAC forces.

Formulation of the SACWARDANS mission requirements began several years ago. It became apparent that the SACWARDANS required capabilities vastly greater than those of previous developments and that the capabilities of existing hardware/software systems would be a limiting factor in the development of the SACWARDANS system. It was hypothesized that some of the new computer architectures, based on newer techniques, might be applicable to the SACWARDANS. It was further thought that inclusion of an associative processing capability within the hardware/software system for SACWARDANS could provide the processing capability required to develop a system to fulfill the SACWARDANS mission requirement. A discussion of this was published by Planning Research Corporation and Informatics Inc. in January 1974 in Report WPO217. This preliminary report led to a more comprehensive

analysis of the applicability of associative processing to the SACWARDANS function. The results of this analysis were published by Planning Research Corporation and Informatics Inc. in June 1974 in Report WPO234. That report contains a summary of the history of on-line data base development, defines associative processing within the SACWARDANS context, presents a model of SACWARDANS processing, describes three candidate hardware systems (two systems with an associative capability and one baseline system without an associative capability), and contains a performance analysis of the three candidate hardware systems with respect to the SACWARDANS system.

This section contains a paraphrase of the SACWARDANS processing model description contained in WPO234. This description has been modified somewhat to clarify it for readers who are unfamiliar with this type of application system. The model remains essentially the same.

The SACWARDANS system is to operate at SAC headquarters. Analysts using the system will be located in secure facilities and will be provided with terminals. These terminals may be either CRT or hardcopy devices. The terminals are connected to an application processor. Currently, there are dual HIS-6080s which may serve as application processors. Initially, it was hypothesized that only one of the processors would service the SACWARDANS application, with the other in use for other purposes. Cutover to the other 6080 would be necessary in the event of system failure. This is a minimal operational requirement--only one processor with switchover in the event of processor failure has access to the SACWARDANS data base at any given time. Implementation of the hardware system in a manner that precludes multiple processor access to the data base is restrictive on system evolution and development. It also precludes functionally shared usage of the associative system. Clearly, sharing the associative system with another application (PACER, for example) is not a functional requirement of SACWARDANS. It does, however, have the potential of enhancing other systems and making wider use of the associative capability.

SACWARDANS has three main functions: 1) accepting input describing the status of the real world, 2) correlating these inputs with historical data to determine status changes, and 3) analyzing historical data to predict future status. The processing model concerns itself with only the first two functions.

2.1.1 Directory Types

In the processing model, the total data base is indexed by several directories. These directories serve as indices for data records which contain additional information. The directories contain most of the data used in analysis. Hence, access to the data records is relatively infrequent, with most of the manipulation being performed on the directories.

The inputs to the system are messages containing status or event reports. An event is defined as an interaction between two or more entities defined as

"organization" positioned at one or more geographic positions defined as "locations" usually involving "equipment" where equipments include people.

Organizations, locations, and equipments are defined to the system. An organization is generally an operating entity. Organization identifiers are encoded to include both a type for the organization and a unique name. Detailed information concerning an organization may be available in the SACWARDANS system in a disk-based record. This record is accessed by an ID which is unique to the organization. The organization IDs serve as keys which are used in a hashing algorithm to access the disk records.

A location has a fixed geographic position, which is located in a country and a geographic area. A uniform system for identifying countries exists. The identifiers are referred to as country codes. There is also a system for identifying geographic areas by area codes. A location is necessarily located in a specific country and a specific geographical area; however, calculation of a country and an area from the location coordinates is not feasible and is unnecessary in a system such as SACWARDANS. It is also the case that the country code does not always uniquely define the geographic area. Locations also have names, a unique ID, and a type--which denotes what type of facility is at the location. The ID serves as a key for use in a hashing algorithm to access an associated disk record for the location.

Equipment records are identified by a name and a unique ID. The ID serves as the access key to a disk record. Each equipment has a type, such as aircraft, and may have a model, such as B52.

The data on organizations, locations, and equipment is relatively static. Clearly, new organizations, locations, and equipment come into existence. Likewise, old ones can cease to exist or to be of interest. However, such changes are infrequent, and there is no requirement to modify the data base on a real-time or near-real-time basis.

Locations define places, equipments define things, and organizations define operational entities. Normally, an organization operates at one or more locations. The organization also has various equipment assigned to it, and the equipment is located at one or more of the locations associated with the organization. Analysts using SACWARDANS require information defining these relationships. Specifically, they need to determine, for each organization, with which locations it is associated, and the types and quantities of equipment it has at each location. Access by location to determine organizations and equipment is also required, as is access by equipments to determine locations and organizations.

The SACWARDANS system receives messages from external sources which report status and events. The volume of messages will vary, reflecting the level of world activity pertinent to the SACWARDANS mission. The messages are retained in the SACWARDANS data base. As messages are received, they are examined by analysts who prepare event records from them. A message may report multiple events, and a single event may be reported in several

messages. An event is an occurrence usually involving organizations, locations, and equipment and occurring at some time or times. The type of occurrence is encoded as an activity code. Also of interest to analysts working with events are geographic area, country code, and environment. All of this information is structured by the analyst, working on-line, into an event record. The event record is added to the data base. The validity of all the IDs (organizations, etc.) and codes should be verified prior to recording in the data base. Since there are multiple messages reporting some events, it is possible that two event records could be prepared for the same event. Hence, the system should attempt to locate potential duplicates. If the system detects any events which may be duplicates, the analyst is requested to resolve the problem.

Messages and events are added to the data base as they are received and constructed. They remain in the active data base for one or more months. Currently, the criterion for retirement of data from systems such as this is antiquity. Usually, a purge algorithm is used which always purges old data after "n" days or which purges the oldest data as space is required for incoming data. Typically, this type of criterion is used because of its simplicity, not necessarily because it is the best. Use of other criteria requires an analysis of what criteria are appropriate and a determination of how to mechanize the application of the criteria. The costs of the analysis and of the mechanization of the choice may preclude its development. The availability of an associative device, plus the research now going on in the adaptation of data base management systems to hierarchical storage systems, may change the costs and thus make it more feasible to use purge criteria other than antiquity. If an antiquity-based purge algorithm is used for SACWARDANS, then it may be necessary to retain messages and events for a period as long as 180 days to perform some types of historical trend analysis.

2.1.2 Processing Operations

As incoming events are received and entered into the data base, they are correlated with previously entered events. These correlations are used by the analysts to identify short-term disturbances to activity patterns. Changes in levels of various types of activity are indicators of arising situations relevant to SAC affairs. Longer-term trend analysis is conceptually similar and is used to identify trends in activity patterns.

In its simplest form, correlation analysis consists of counting the number of events meeting specified criteria within a span of time. An example would be to count all those events which occurred within the same hour, the same environment, and the geographic area and with the same activity code as that of the event being correlated. This correlation may be modified by requesting a count of the number of equipments involved in the identified events. For example, the first correlation would count the number of events involving aircraft takeoffs, whereas the second correlation would count the number of aircraft taking off. The results of these counts may be compared with threshold limits which trigger reports to analysts and/or other actions.

As new events are added to the data base, the application program selects a predefined sequence of correlations which is to be performed. The selection process is a function of the type of activity. The sequence could contain decision points (e.g., branches) which depend on values determined up to that point. For example, additional correlations might be performed if initial correlations result in values outside of threshold limits. Analysts may develop special sequences of searches which are evoked automatically or which may be retained for execution upon request--that is, their execution is not triggered by creation of a new event. These special sequences are similar to those triggered by addition of events.

The correlation procedures currently in use are fairly simple sets of searches, with the resulting values being compared with threshold limits. This is done by using summary tables containing running counts of various occurrences. This type of mechanization is feasible on existing hardware but does not offer the option of performing ad hoc searches. Current searches are certainly constrained by the available computing equipment.

It would seem desirable to be able to extend the correlation concept to examine the trend. This may be done by a succession of searches spanning several intervals of time. Results of this type could be returned to the analyst in numerical form or in a graphic display.

2.2 SACWARDANS DIRECTORIES DATA BASE

The SACWARDANS directories require processing which can be easily done on a system with an associative capability. This fact, coupled with the response requirements and extent of processing, has resulted in development of a processing model in which the directories are stored in associative storage. The longer records which the directories index are retained on disk from which they may be accessed when required. The directories include sufficient data to permit most trend and correlation analysis to be performed against the directories without reference to disk-stored data.

This section describes the content and non-overhead storage requirements of the SACWARDANS directories data base. The content is described in a relational form as a system user would perceive it. (A non-associative implementation would require some type of structural overhead to reduce brute-force searching.) An associative implementation uses representation similar to the relational form used in the description, but with some overhead for record identification, work space, and so forth.

2.2.1 Directory Content

The data content and record organization for the SACWARDANS directory data base was previously described in PRC/Informatics' Report WPO234. This description has been reviewed. There are some modifications which have been incorporated in the description provided here. These changes are minor and

do not alter the concepts--only details. The description of the data base should be viewed as typifying the SACWARDANS directory data base requirement, not as an exact description of the data base content and record organization.

There are six indexes in the directory data base. They are:

- Equipment index,
- Organization Index,
- Location Index,
- Message Index,
- Event Correlation Index
- Location/Organization/Equipment Correlation Index.

Each of these is described and discussed in the paragraphs that follow.

2.2.1.1 Equipment Index -- The equipment records describe kinds of equipment which are relevant to the SACWARDANS mission. The equipment records, which are stored on disk, are indexed by the Equipment Index. It is anticipated that the Equipment Index would be stored in the ECAM. The index contains four items: equipment name, type, model, and an equipment ID. The name is an alphanumeric name of up to 18 characters. Alphanumerics require six bits per character because the computer on which the application programs operate uses a six-bit representation. It is likely that the representation will eventually change to eight bits per character. The type describes the kind of equipment (e.g., aircraft), and the model defines a model within type (e.g., B52). The equipment ID is a unique identifier, which, when used as a key in a hashing algorithm, results in an access address for the equipment record. Table 3 lists the items in the Equipment Index and the number of bits each is expected to require,

Table 3. Equipment Index

Content	Bits Required
Equipment name	108
Type	36
Model	36
Equipment ID	36
Total	216

2.2.1.2 Organization Index -- The organization records contain data defining organizations. Organizations are operational entities which may have equipment assigned to them, be assigned to one or more locations, and participate in events. The organization records are listed in the Organization Index. The index contains the organization name, a unique ID, and a type. The name is alphanumeric with up to 38 characters. The ID is transformed via a hashing algorithm and is then used to access the corresponding disk record. The content of the Organization Index and its anticipated bit requirements are shown in Table 4.

Table 4. Organization Index

Content	Bit Required
Organization name	228
Organization ID	36
Organization type	18
Total	282

2.2.1.3 Location Index -- The location records describe places. These may, for example, be air bases. The records, which are stored on disk, are indexed by the Location Index. Each entry in the Location Index contains the name of the location (up to 38 alphanumeric characters) and a unique ID. As is the case with the other indexes, the location ID serves as a key which, after transformation in a hashing algorithm, serves as the relative address of the corresponding record on disk. The Location Index also specifies the type of facilities at the location.

The remaining items define where the location is. The coordinates define its location on the earth. The country code specifies the country in which it is located. The geographic area specifies the area in which it is located. The coordinates uniquely determine the country and area. However, the conversion from coordinates to country or area is not simple. A country may be split with parts in different areas. Areas usually span many countries or parts of countries.

Table 5 lists the content and anticipated bit assignments for the Location Index.

2.2.1.4 Message Index -- The Message Index lists the messages which are received and, after processing, stored on disk. Each message has two numbers assigned to it. One number is assigned by the originator and is transmitted with the message. The second number serves as a unique ID which is used as a key that is transformed into a disk address for access to the message. Messages also have two times associated with them. The first time--date, time, group (DTG)--specifies when the message was sent; the second time specifies when it was received. The remaining items specify who sent the message and its precedence. Table 6 lists the items and anticipated bit assignments.

Table 5. Location Index

Content	Bits Required
Location name	228
Coordinates	90
Type	30
Location No. ID	20
Location country code	12
Location geographic area	18
Total	398

Table 6. Message Index

Content	Bits Required
Message No. ID	20
Receipt DTG	40
Message precedence	12
Originator ID	72
Transmit DTG	40
Originator message No.	22
Total	206

2.2.1.5 Event Correlation Index -- The Event Correlation Index entries are created by analysts working at consoles. Thus, these entries do not index disk records but rather record events that have occurred. More extensive event data could, however, be structured into records which are stored on disk and accessed by key. The Event Correlation Index entries refer to messages, locations, organizations, equipments, and times. The messages are those which reported the event. The locations, organizations, and equipment are those connected with the event. The times are start and stop date time groups (DTG).

The remaining items specify activity, environment, country code, and geographic area. The activity refers to the type of event. Environment, country code, and geographic area are items used as terms in correlation analysis. Table 7 lists the Event Correlation Index items.

Table 7. Event Correlation Index

Content	Bits Required
Event No. ID	20
First message No. ID	20
This message No. ID	20
Start DTG	40
Stop DTG	40
Start location No. ID	20
Stop location No. ID	20
Equipment ID	36
Equipment count	12
Initiating organization ID	36
Receiving organization ID	36
Environment	6
Activity	6
Geographic area	6
Country code	12
Total	330

2.2.1.6 Location/Organization/Equipment (L/O/E) Correlation Index -- The L/O/E Correlation Index contains an entry for each type of equipment assigned to an organization at a specific location. A location may have multiple organizations assigned to it. Likewise, an organization may be assigned to multiple locations. Each organization may have several types of equipment dispersed in various ways to the associated locations. The L/O/E defines all of these relationships, including the quantities of equipment involved. Table 8 lists the content of the L/O/E Correlation Index.

Table 8. L/O/E Correlation Index

Content	Bits Required
Location No. ID	20
Organization ID	36
Equipment ID	36
Count	12
Total	104

2.2.2 Directory Sizes

The size of the directories, exclusive of overhead, has been estimated and is shown in Table 9. The number of entries of each type has been retained from the prior study. Deviations from these values are probable, but these are reasonable estimates. The number of messages and events is based on an average of 3000 messages per day, with an average of one event per message, and retention for 180 days in the ECAM array. The size of the entries is based on the record contents shown in Tables 3 through 8. Record sizes are rounded up to eight-bit bytes. Eight-bit bytes are used as the unit of measure because discussions of ECAM architecture have used the eight-bit type. This should not be confused with the use of six bits for encoding of alphanumeric data.

2.3 DIRECTORY ALGORITHMS

The SACWARDANS application requires algorithms for maintaining the directories data base, for performance of trend and correlation analysis, for storage management, and for restoration of the directories data base in the event of failure. Maintenance of the directories data base refers to the addition, modification, and deletion of directory entries. This represents a significant component in system workload, with most of that component being maintenance of the message and event indexes. Trend and correlation analysis supports the near-real-time analysis activities of the SACWARDANS analysts. This is the largest single component of system activity, overshadowing all others. Storage management consists of those functions which are typically called system housekeeping: initializing formats, reapportioning the allocation of space to

Table 9. SACWARDANS Data Base Size (Bytes of Data Only)

Directory	Entry Size, Bytes (8 bits/byte)	Number of Entries	Total Bytes (8 bits/byte)
Equipment Index	27	1,000	27,000
Organization Index	36	5,000	180,000
Message Index	26	540,000	14,040,000
Location Index	50	5,000	250,000
L/O/E Correlation Index	13	30,000	390,000
Event Correlation Index	42	540,000	22,680,000
Total Data	194	1,121,000	37,567,000

different types of entries, and so forth. This represents a very small portion of workload and is one which normally occurs when the system is not busy with other tasks. Restoration of the directories data base must be done rapidly. This infers that it cannot be done by reprocessing of disk-stored records. Hence, a backup to the associative storage must be provided by hardware/software means.

This subsection contains flowcharts and explanatory text for algorithms required for maintaining and using the SACWARDANS directory data base. These are functionally specified, but modifications resulting from particular mappings of the data base into the associative storage are not presented, as they are handled within the slave.

2.3.1 Maintenance of the Directories Data Base

The SACWARDANS application requires algorithms for posting new entries into the indexes, deleting entries no longer required, and modifying existing entries.

The L/O/E indexes are relatively static. Hence, they do not impose any significant workload. The algorithms for adding, deleting, and modifying them are simple and not time-consuming. The L/O/E Correlation Index is also not expected to impose much of a workload, and, again, the algorithms are relatively simple.

The Message and Event Correlation Indexes require extensive update activity each day. The volume of new entries is such that a purge procedure must be operated on a regular basis so as to avoid memory saturation. Further, each event entry will trigger execution of standard correlation analysis procedures. Hence, the total activity associated with message and event processing is substantial.

2.3.1.1 Maintenance of L/O/E and L/O/E Correlation Indexes -- The algorithms for maintaining (adding, modifying, and deleting entries) the L/O/E and L/O/E Correlation Indexes are simple, similar to each other, and infrequently executed. Flowcharts, at a functional level, are provided in this subsection. The flowcharts also show which major system component(s) are expected to perform each function of the algorithm. The flowcharts are further explained in the text. Because the algorithms are similar for each of the directories, the explanatory text in each case discusses aspects of the algorithm which differ from previous explanations.

2.3.1.1.1 Maintenance of the Location Index -- After the Location Index has been initially loaded, it is necessary to be able to add new entries, delete obsolete entries, and modify existing entries. The add new entries procedure can also be used for initial loading of the index.

Figure 8 is a functional flowchart for Add Location. The steps are as follows:

- 1) The entry to be added is constructed by an analyst or perhaps a designated system manager or data base administrator. The originator may directly prepare the Location Index entry or may prepare a longer record which is to be stored on disk. If the originator prepares a longer record, an application program will subset it to prepare the index entry.
- 2) The application program receives the entry from the originator and assigns a unique ID to the entry.

The next step is to edit the entry to verify its consistency and to determine that it does not duplicate any other entry in the Location Index. The application program specifies what checks are to be made and directs the searches of the Location Index and several code lists. The associative system performs the searches. The edit procedure is mostly self-explanatory. Duplicate names, coordinates, and ID are not allowed. The remaining edit checks validate that the encodements for location type, country code, and geographic area are valid codes. The number of entries in these code lists (as well as several other similar lists associated with other indexes) is small. There may be as few as five to 10 entries or, in the case of country code, a few hundred. These code validation edits are easily mechanized within the application program but are naturals for associative search. Hence, the code lists will undoubtedly be recorded in the associative storage in otherwise wasted space.

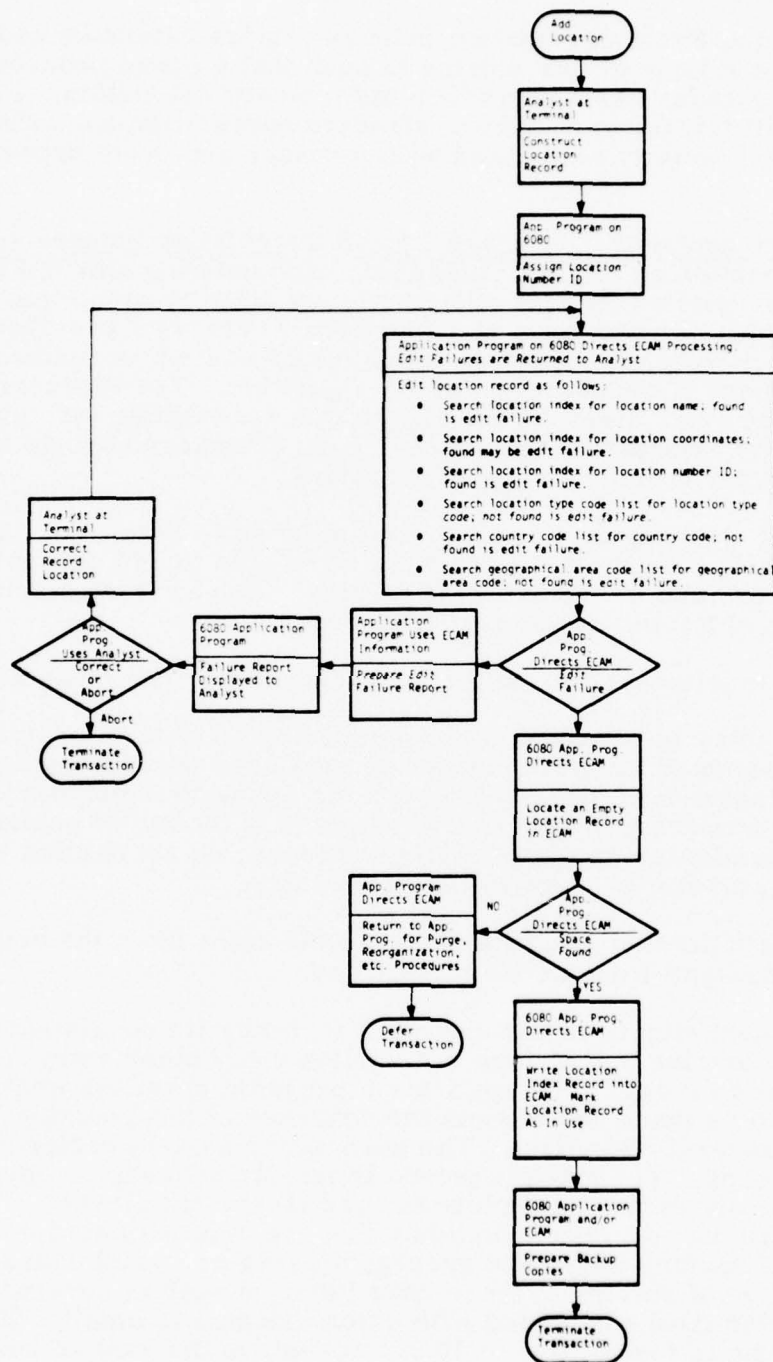


Figure 8. Add Location

- 3) If any of the edit checks fail, then the originator is notified. The report sent back must indicate which edit checks failed. The originator may abort the entry or correct it.
- 4) If the edit is successfully completed, the application program directs the ECAM to locate an empty space assigned to the Location Index and then enter the entry. The marking which indicates the busy/not busy state of the space must also be updated.
- 5) Finally, a backup copy must be recorded for use in the event of system failure.

Figure 9 is a functional flowchart for Modify Location. The requestor specifies the ID of the location to be modified. The application program directs the ECAM to search the Location Index for the location and to return the entry to the application program. The entry is then displayed to the requestor who enters the modifications. The modification is then recorded in the Location Index by executing the algorithms for Delete Location and Add Location.

Figure 10 is a functional flowchart for Delete Location. The requestor specifies the ID of the location to be deleted. The application program directs the ECAM to search the Location Index for the location with that ID. If it is found, the entry is deleted and the space occupied by it marked not busy. The deletion is logged on the backup copy record.

2.3.1.1.2 Maintenance of the Organization Index -- After the Organization Index has been initially loaded, it is necessary to be able to add new entries, delete obsolete entries, and modify existing entries. The add new entries procedure can also be used for initial loading of the index.

Figure 11 is a functional flowchart for Add Organization. The operation of Add Organization is structurally similar to that of Add Location. Only the specifics of the edit checks vary.

Figures 12 and 13 are functional flowcharts for Modify Organization and Delete Organization, respectively. These algorithms are essentially identical to the corresponding algorithms for the Location Index.

2.3.1.1.3 Maintenance of the Equipment Index -- After the Equipment Index has been initially loaded, it is necessary to be able to add new entries, delete obsolete entries, and modify existing entries. The add new entries procedure can also be used for initial loading of the index.

Figure 14 is a functional flowchart for Add Equipment. The operation of Add Equipment is structurally similar to that of Add Location. Only the specifics of the edit checks vary.

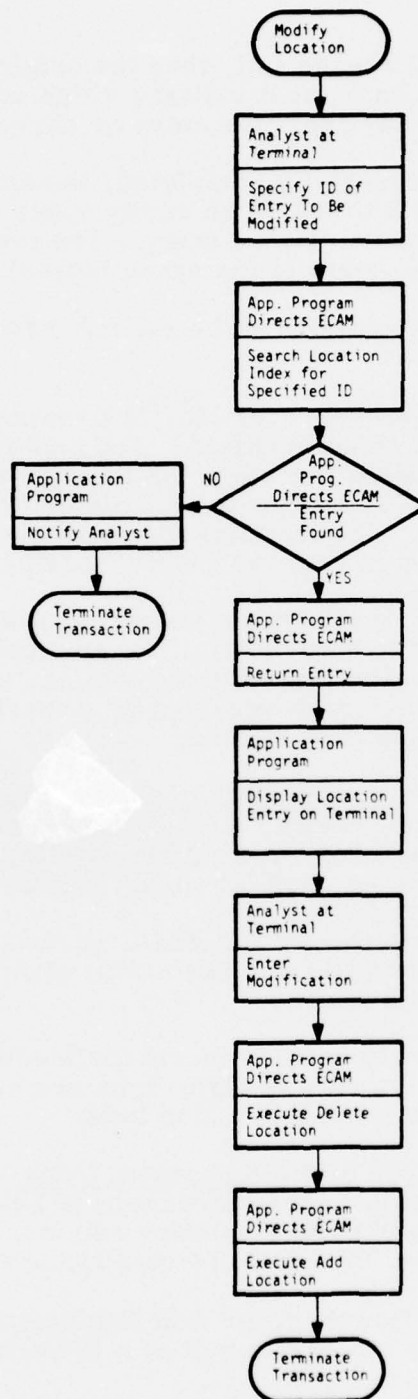


Figure 9. Modify Location

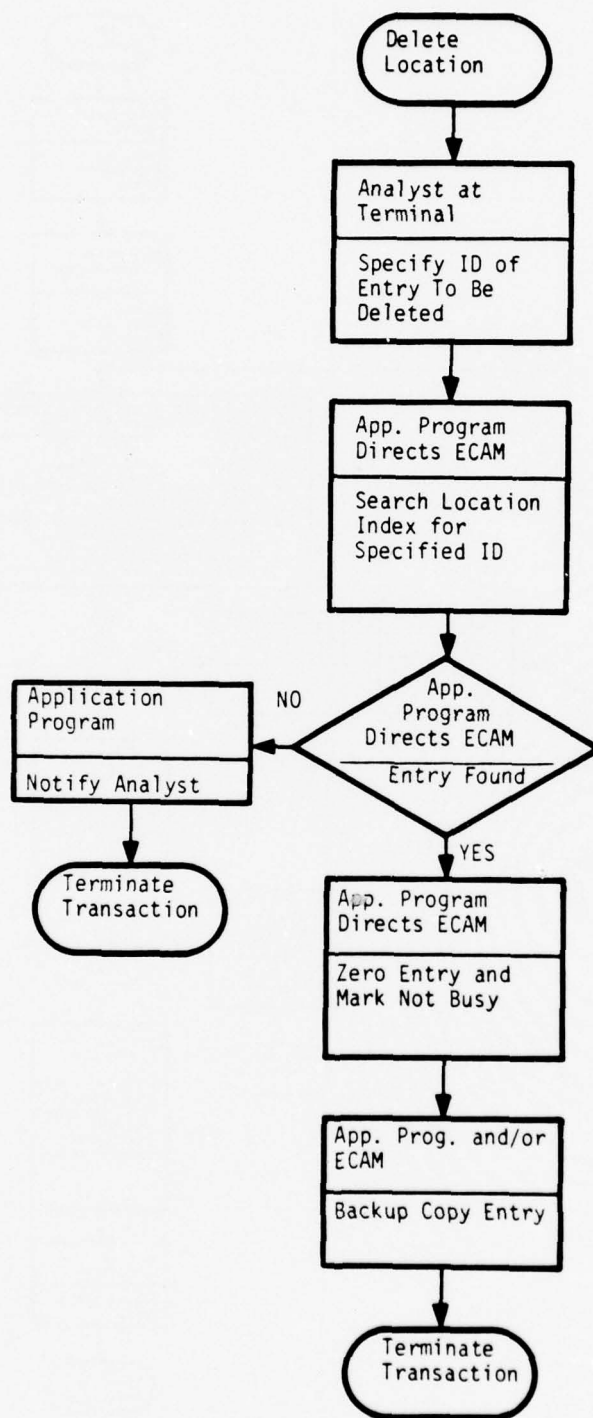


Figure 10. Delete Location

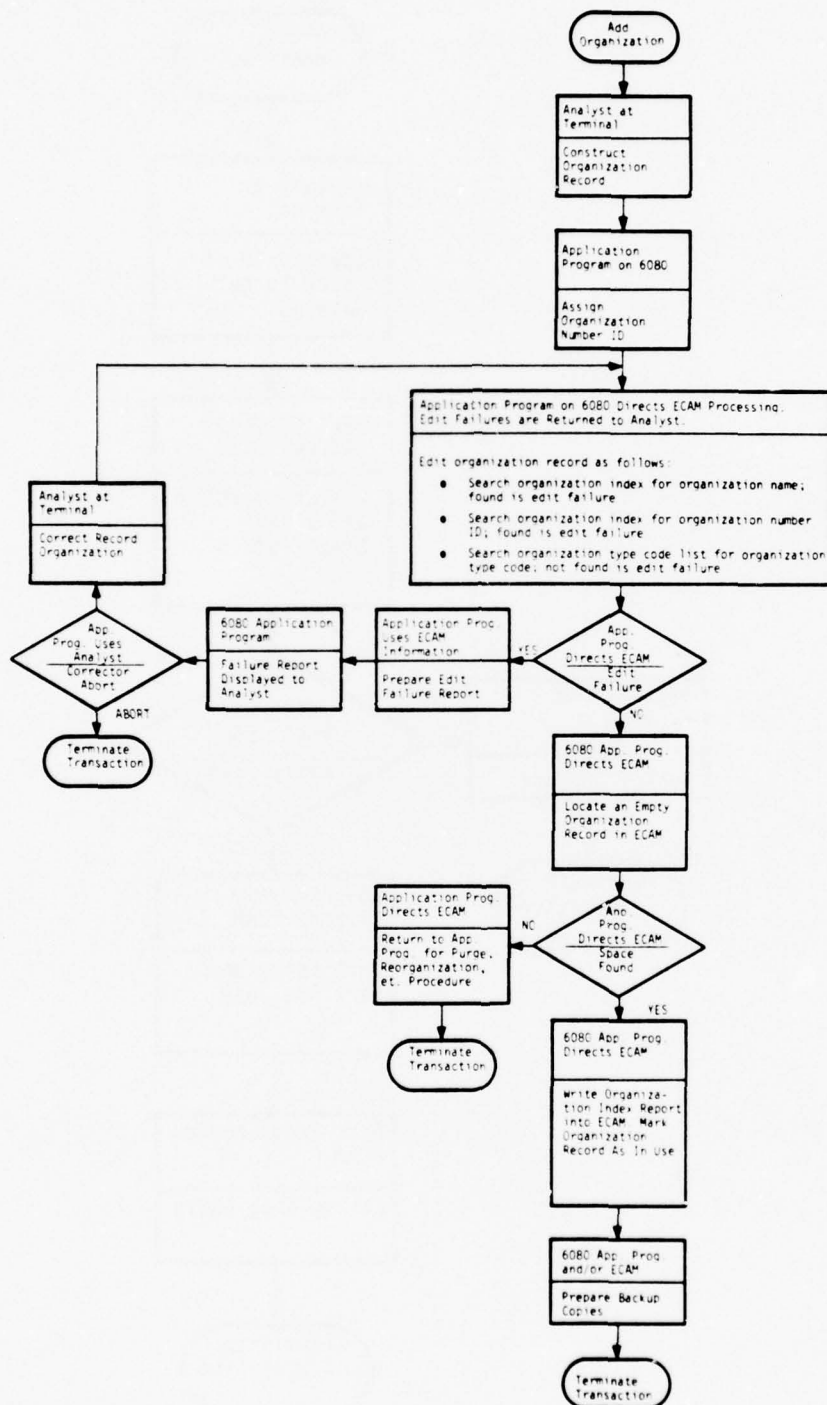


Figure 11. Add Organization

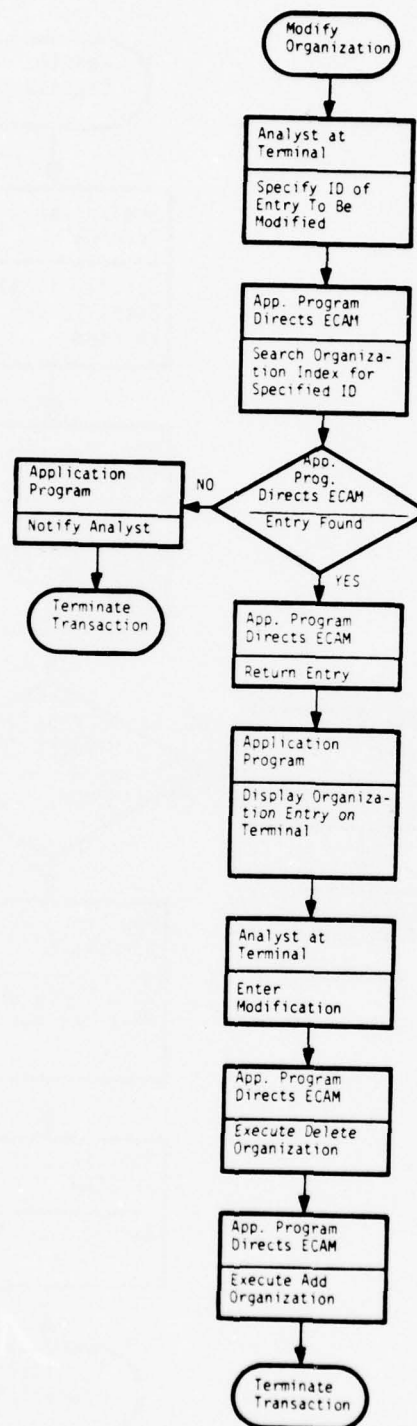


Figure 12. Modify Organization

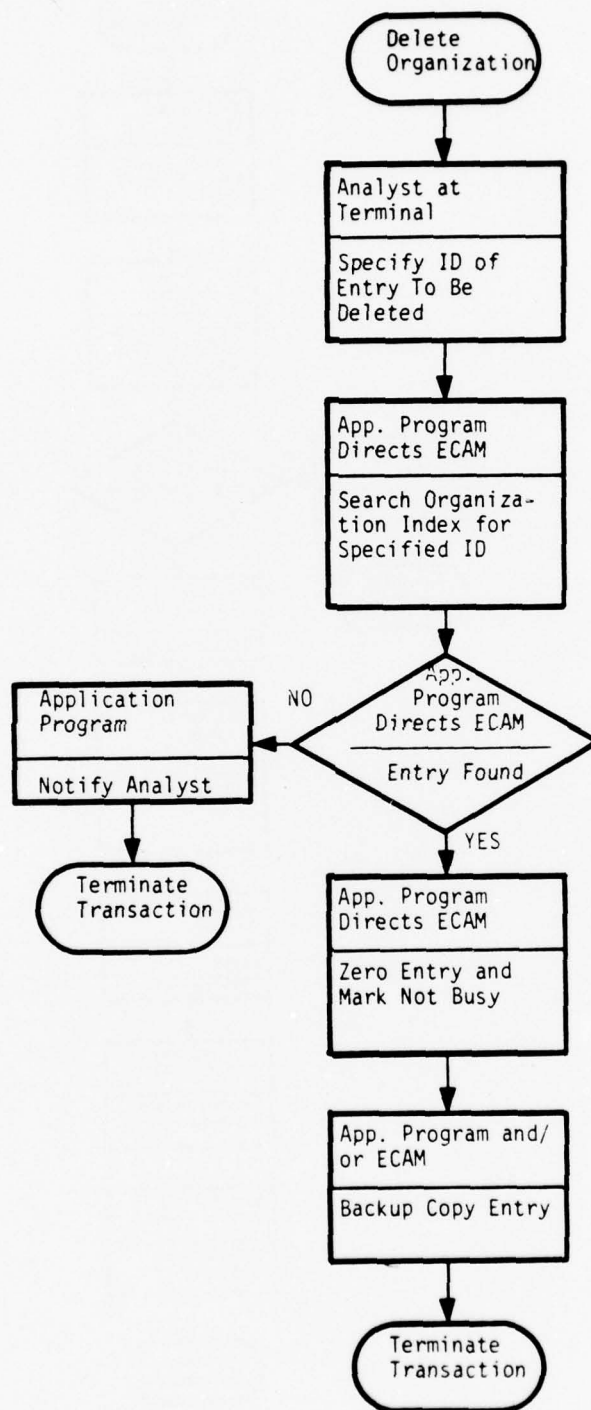


Figure 13. Delete Organization

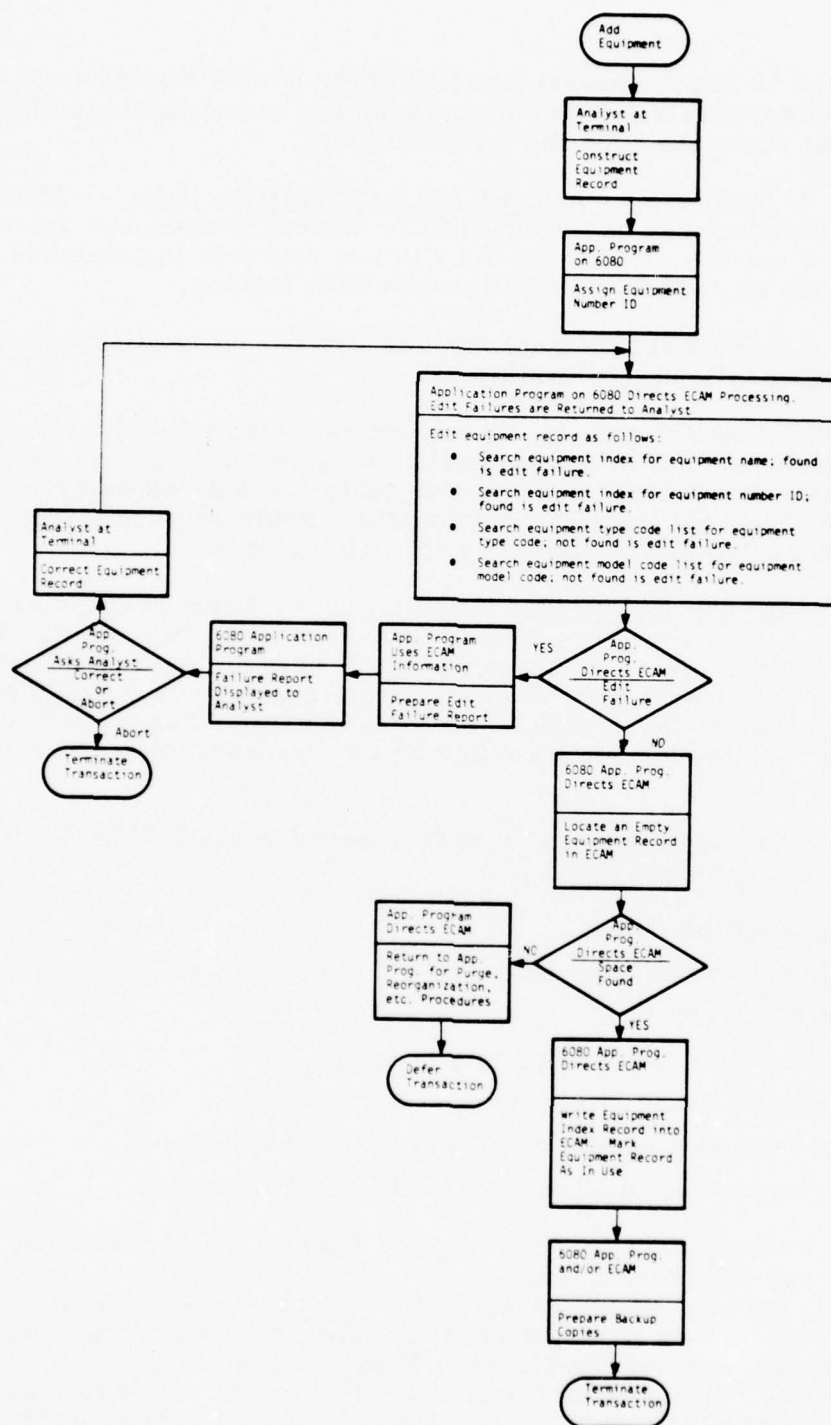


Figure 14. Add Equipment

Figures 15 and 16 are functional flowcharts for Modify Equipment and Delete Equipment, respectively. These algorithms are essentially identical to the corresponding algorithms for the Location Index.

2.3.1.1.4 Maintenance of the L/O/E Correlation Index -- After the L/O/E Correlation Index has been initially loaded, procedures are required for adding new entries, modifying old entries, and deleting obsolete entries. These may also be used for the initial data base loading.

Figure 17 is a functional flowchart for Add L/O/E. It is similar to those for locations, organizations, and equipment.

Figures 18 and 19 are functional flowcharts for Modify L/O/E and Delete L/O/E, respectively. These are similar to those for locations, organizations, and equipment, except that the requested entry can only be uniquely identified by a combination of the three IDs for location, equipment, and organization. Hence, all three must be specified by the originator.

2.3.1.2 Message and Event Index Maintenance -- Event records are created by analysts at terminals. As messages are received, they are displayed to an appropriate analyst. The analyst may review several messages before an event is formulated. It is also possible for a single message or a group of related messages to result in the creation of several events. The process of creating an event is an application program function which does not require ECAM processing.

When an event has been created, it is then added to the ECAM Event Index. The steps in this process are:

- 1) Construct the event.
- 2) Assign event number ID.
- 3) Edit the event.
- 4) Search for possible duplicate events.
- 5) Locate an empty Event Index record.
- 6) Write the Event Index record in the ECAM array.
- 7) Prepare backup copies of record for recovery use.

Figure 20 is a functional flowchart for Add Event. All of the steps, except for those associated with duplicates, are similar to those used in the add procedures for locations, organizations, and equipment. An analyst can construct a duplicate event if it is reported several times by incoming messages. The system can detect possible duplicates but cannot positively identify them as actual duplicates. The search for possible duplicates requires locating any events that occurred at approximately the same time, involving the same locations and organizations in the same type of activity with the same type of equipment. This involves a simple search of several fields in the Event Correlation

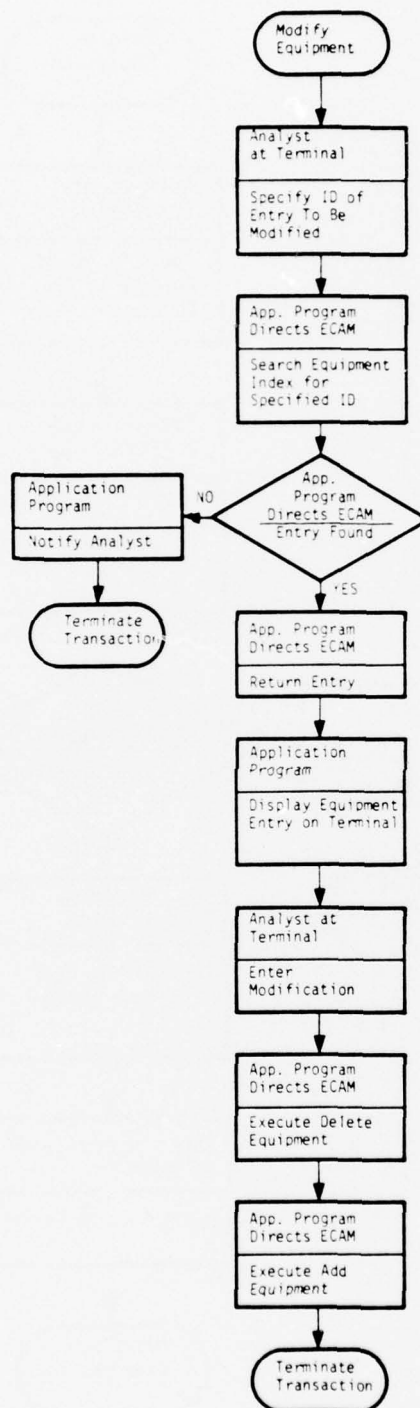


Figure 15. Modify Equipment

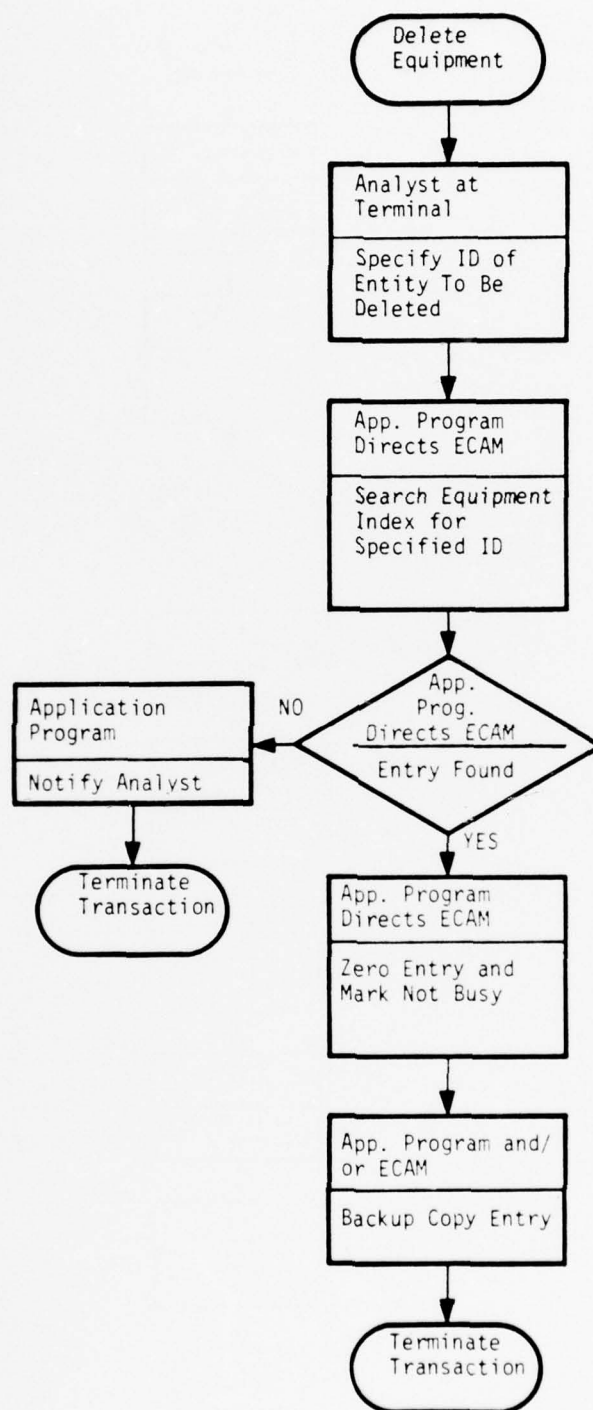


Figure 16. Delete Equipment

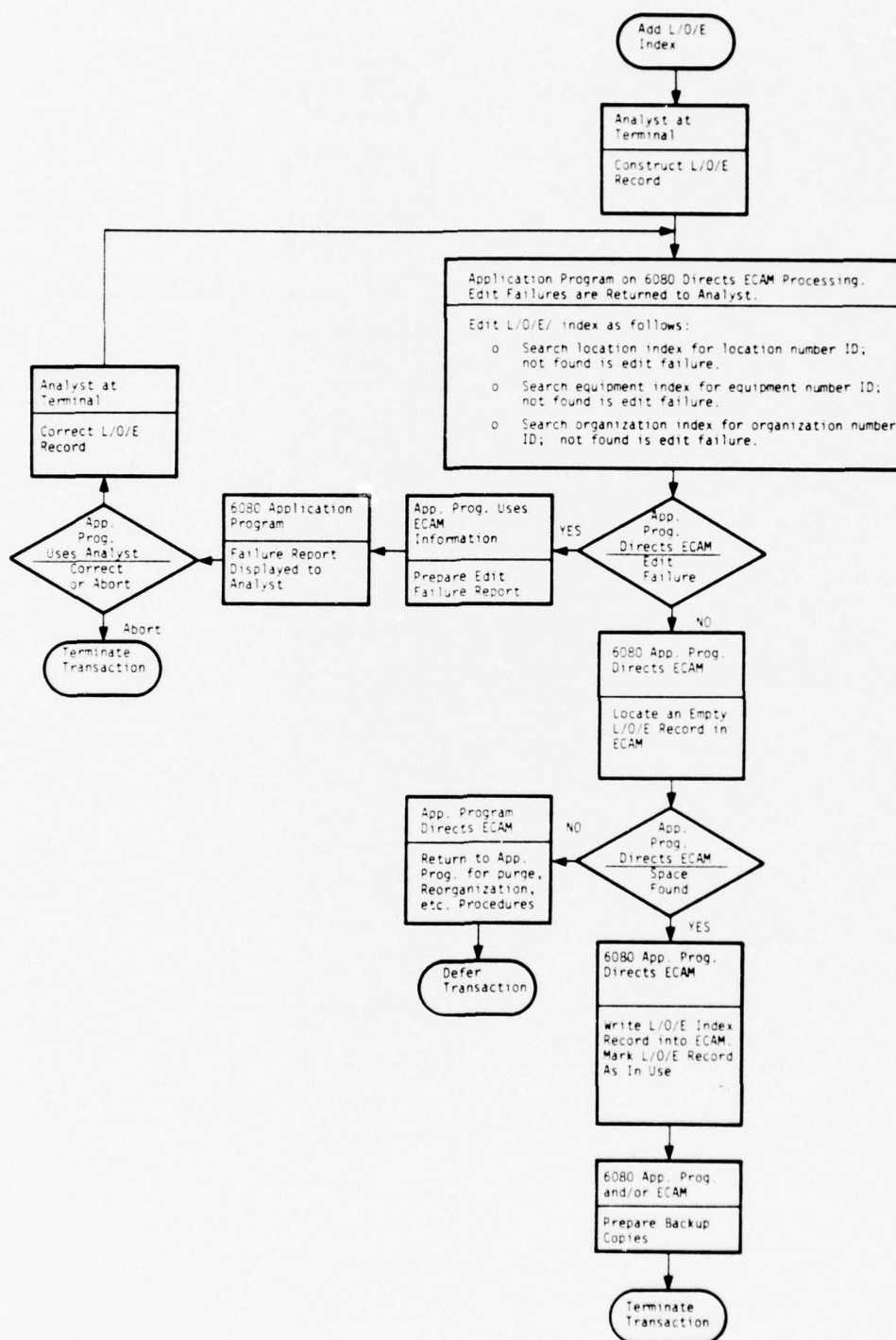


Figure 17. Add Location/Organization/Equipment

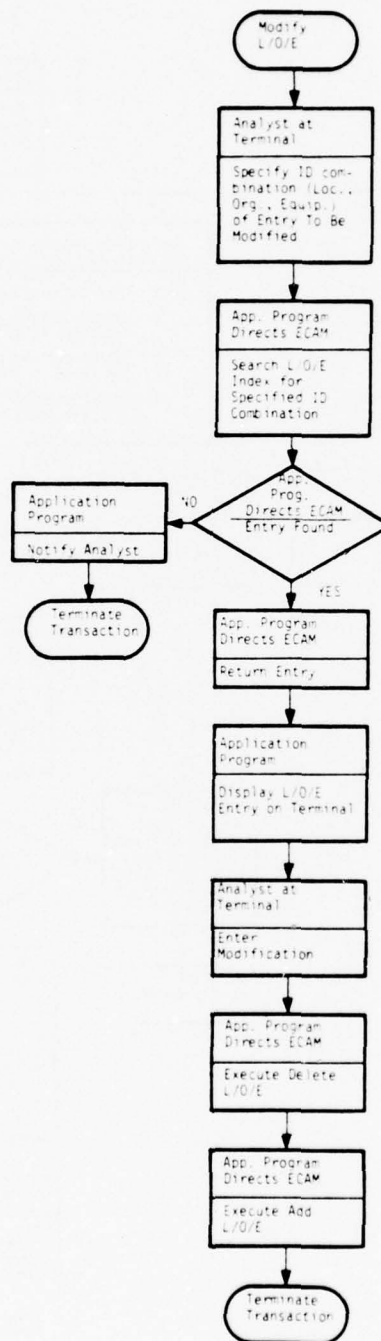


Figure 18. Modify Location/
Organization/
Equipment

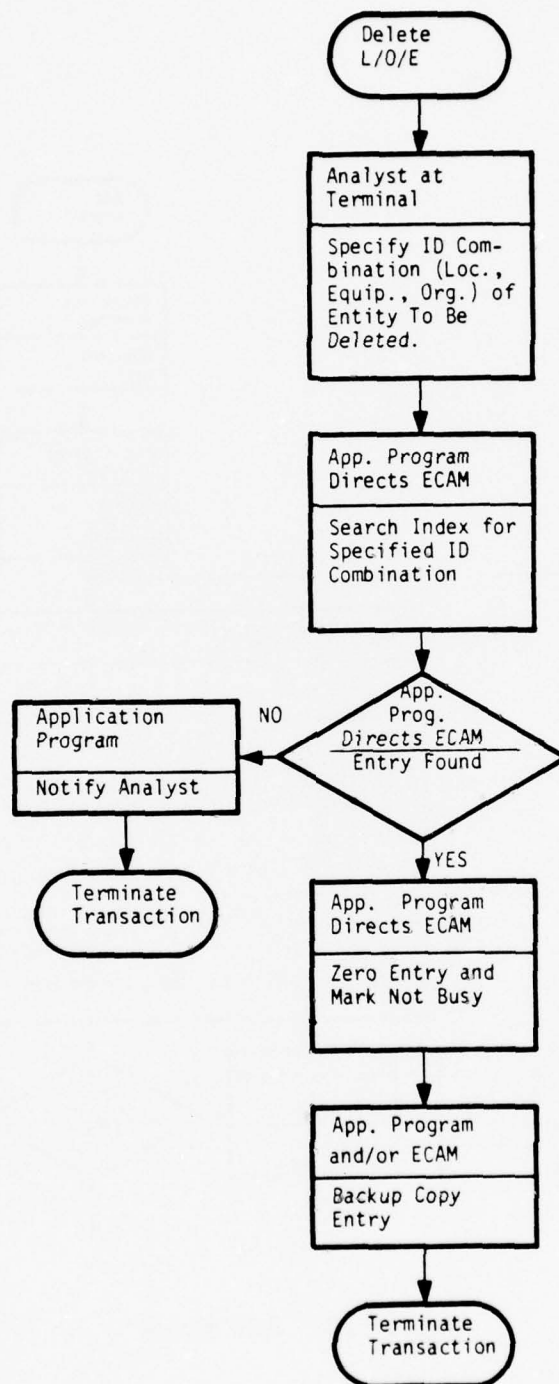


Figure 19. Delete Location/Organization/Equipment

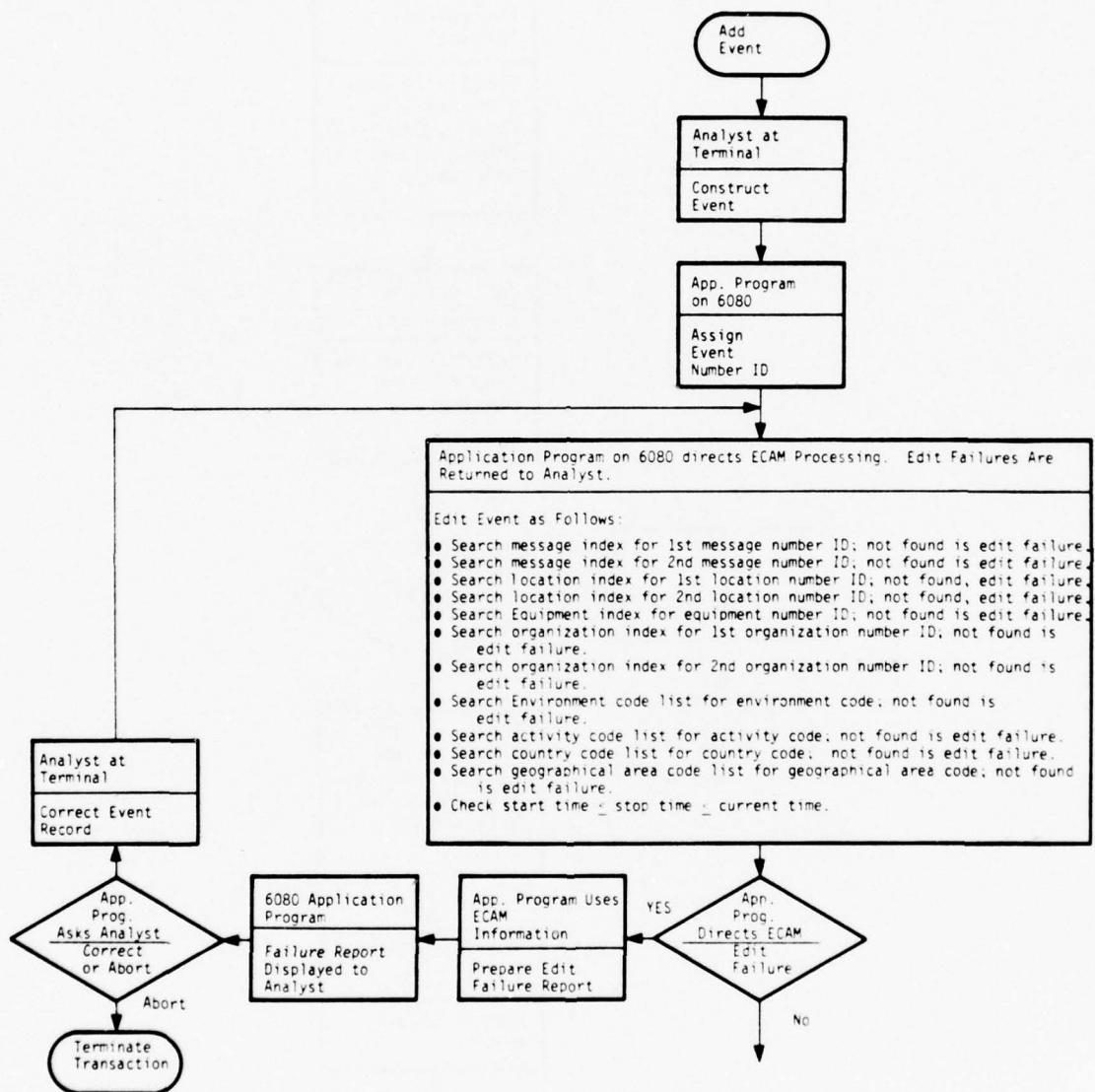


Figure 20. Add Event

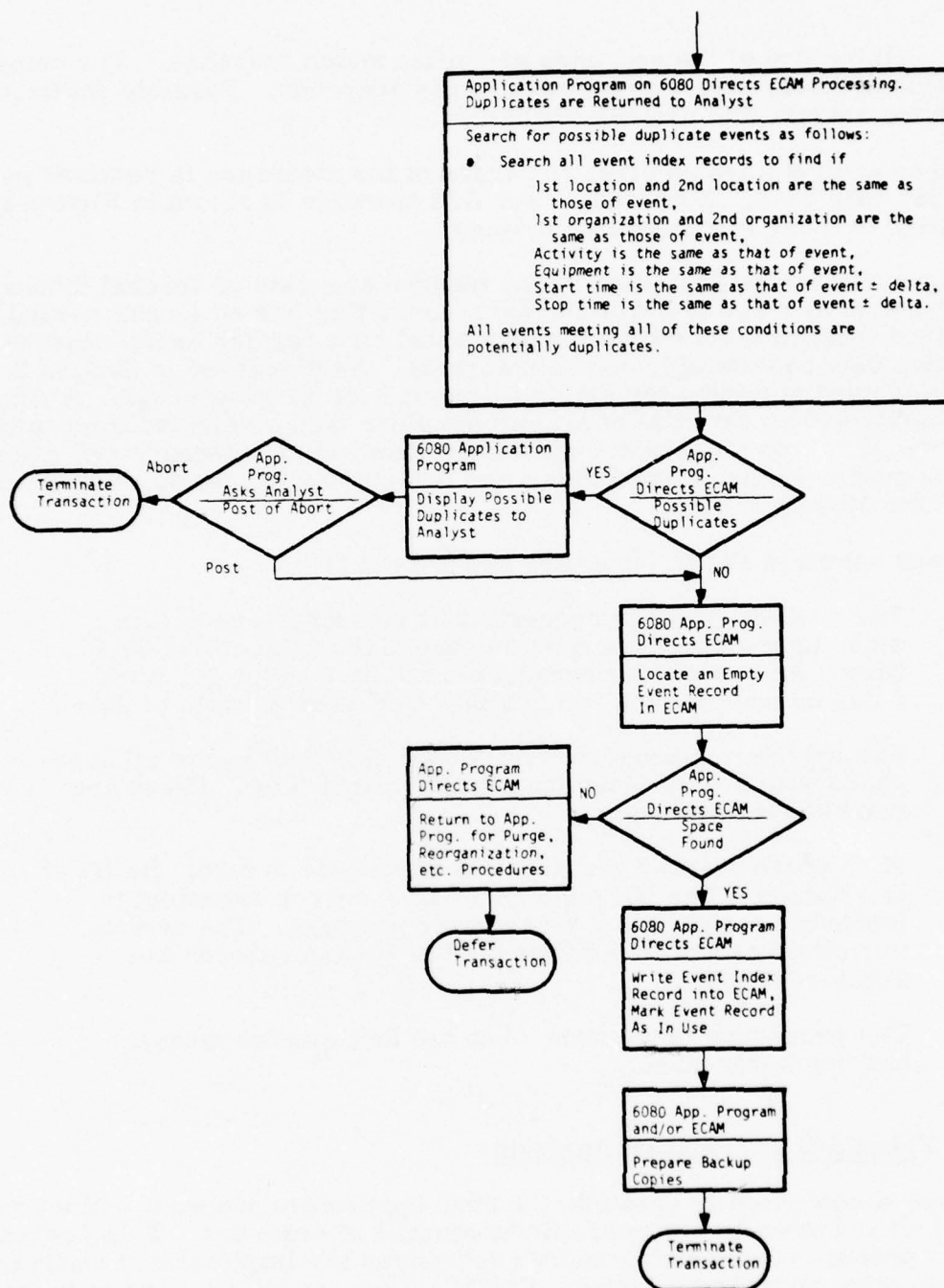


Figure 20. Add Event (Concluded)

Index. All but two of the searches are exact match searches. The remaining two fields are searched with between limits searches. Possible duplicates are sent to the analyst who decides what to do.

Messages are retained on disk. An index of the messages is retained in the directory data base. A procedure for Add Message is shown in Figure 21. It is similar to those for the other indexes.

Messages and events are added to the indexes at a rate of several thousand (an estimated average of 3000) per day. Those that are no longer needed must be retired (purged from the active data base) on a regular basis; otherwise, the active data base would become saturated. As discussed in Section 2.1, the commonly used criterion for selecting entries for purging is age. A purge algorithm based on removal of all entries older than a selected time is shown in Figure 22. This algorithm coordinates removal of messages and events so that any events in the Event Correlation Index refer to messages which are still in the Message Index.

The steps shown in Purge Messages and Events are:

- 1) The system manager requests that messages and events older than a specified time be purged from the active data base. An automatic procedure could be used (e.g., once a day or once a week purge a day's or week's worth of data).
- 2) The application program directs the ECAM to locate all messages with a DTG older than the specified time. These are marked for retrieval.
- 3) Each of the marked messages is processed in turn. Its ID is obtained. The ID is then used as a search argument to locate all events which refer to the message. The events and message are then deleted. The backup records are similarly updated.
- 4) The procedure terminates when the last marked message has been processed.

2.3.2 Trend and Correlation Analysis

Whenever a new event is created, the 6080 application program will automatically select and execute a prespecified sequence of searches. This process and also the process of specifying search sequences are implemented entirely in the application programs. Hence, ECAM activity is not involved in these functions.

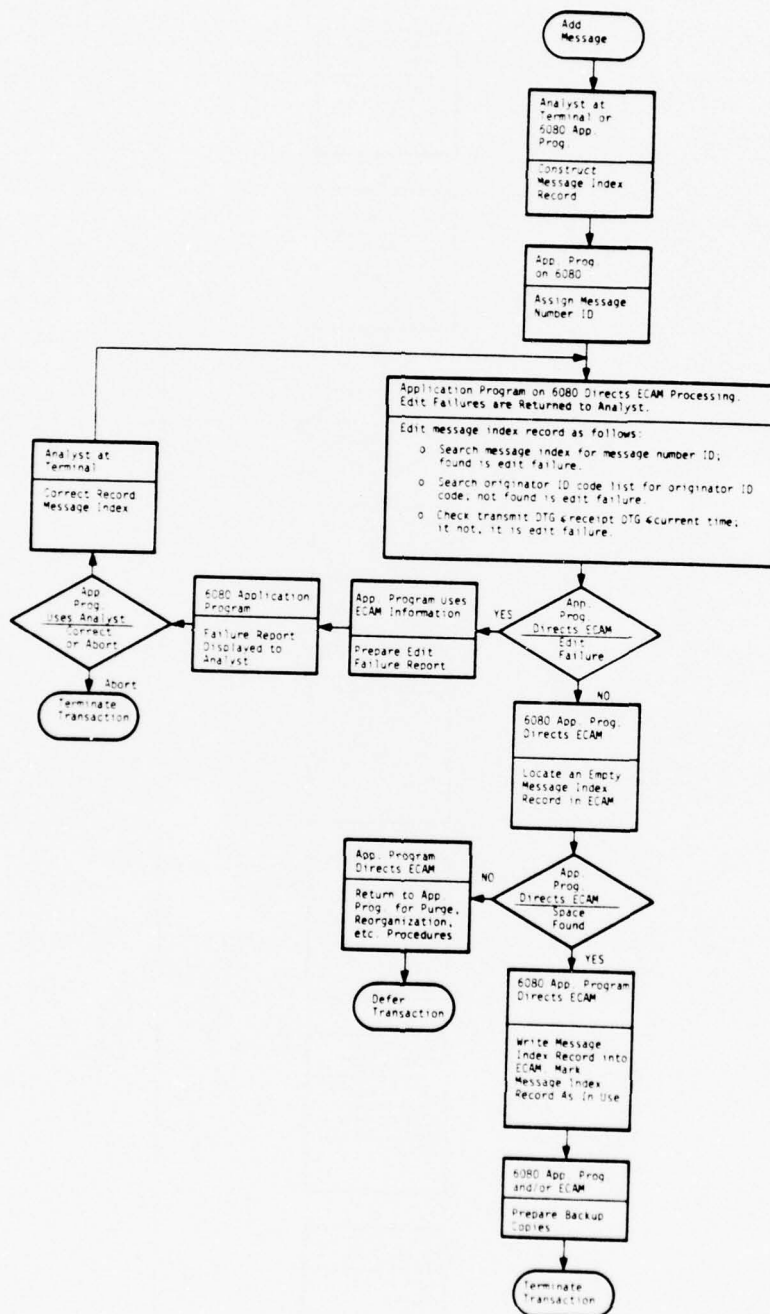


Figure 21. Add Message

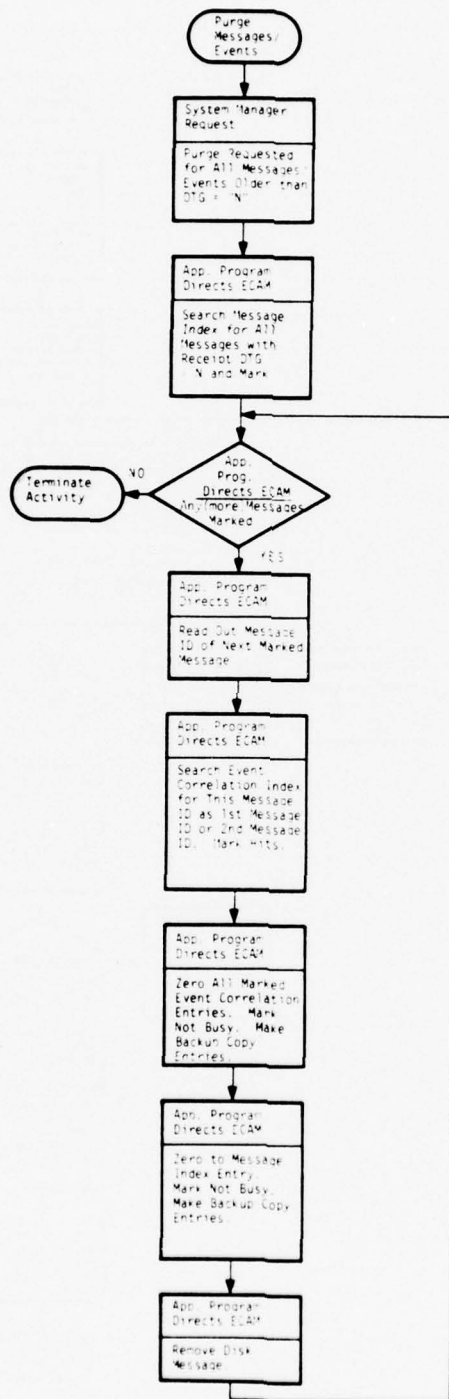


Figure 22. Purge Messages and Events

Analysts may develop special sequences of searches which are evoked immediately or which may be retained for execution upon request; that is, their execution is not triggered by creation of a new event. These special sequences are similar to those triggered by addition of events.

The correlation procedures currently in use for SACWARDANS are fairly simple sets of searches, with the resulting values being compared with threshold limits. It appears likely, however, that more elaborate procedures might be developed once the ECAM is available and the analysts become aware of the potential power of the device. Current procedures are certainly constrained by the available computing equipment.

The current correlation procedures use summary tables containing counts of various occurrences. This is not the most suitable approach for ECAM implementation.

The following are examples of correlation queries:

- How many events with activity-type "value" have occurred in the last "value" hours in geographic area "value" and with country code "value" and in environment "value?"
- Is the number of events with activity-type "value" which have occurred in the last "value" hours in geographic area "value" and with country code "value" and in environmental "value" greater than the threshold "value?"
- How many events with activity-type "value" or "value" or "value" have occurred in the last "value" hours in geographic area "value" or "value" and in an environment between "value" and "value?"

Figures 23, 24, and 25 are functional flowcharts of these three correlation queries.

It is estimated that typical events would trigger a prespecified series of correlation queries containing approximately 13 queries, each of which is about as complex as those listed above.

The above-mentioned queries give typical search sequences. They do not fully represent the types of responses the analysts want to receive. The response patterns which have been identified are:

- How many responders are there?
- Does the number of responders exceed a threshold "value?"
- How many items of equipment are referenced in the responders? (This is the sum of the equipment count field over all responders.)

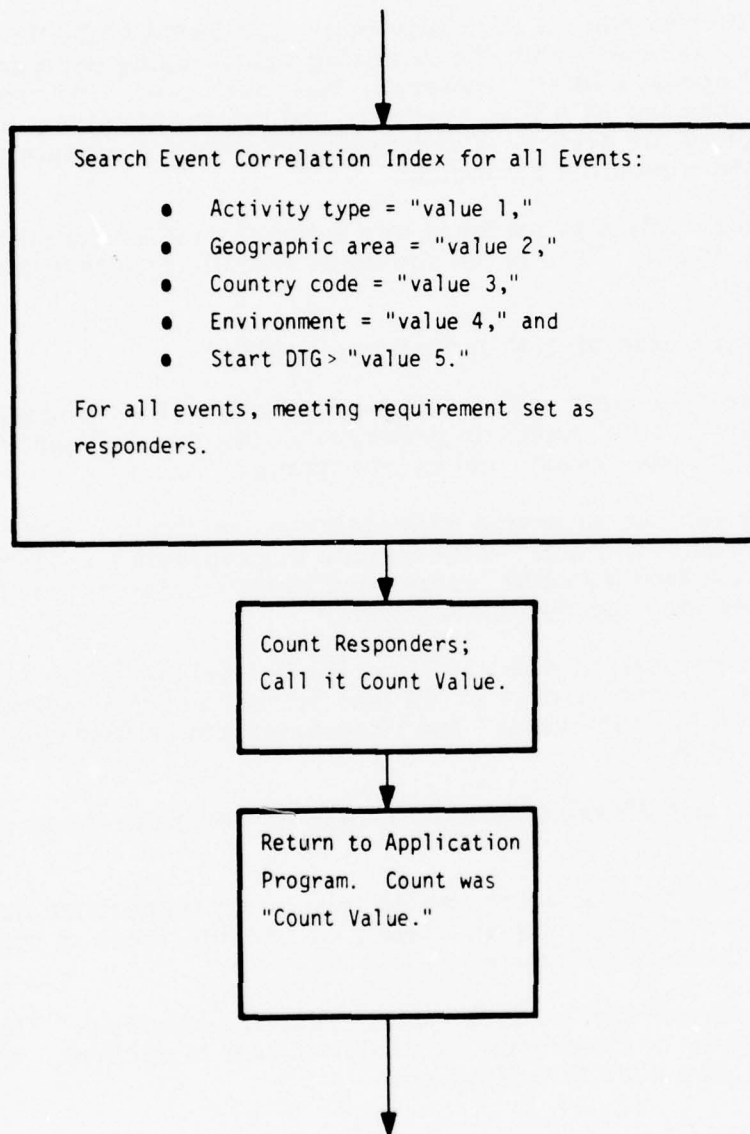


Figure 23. First Sample Query

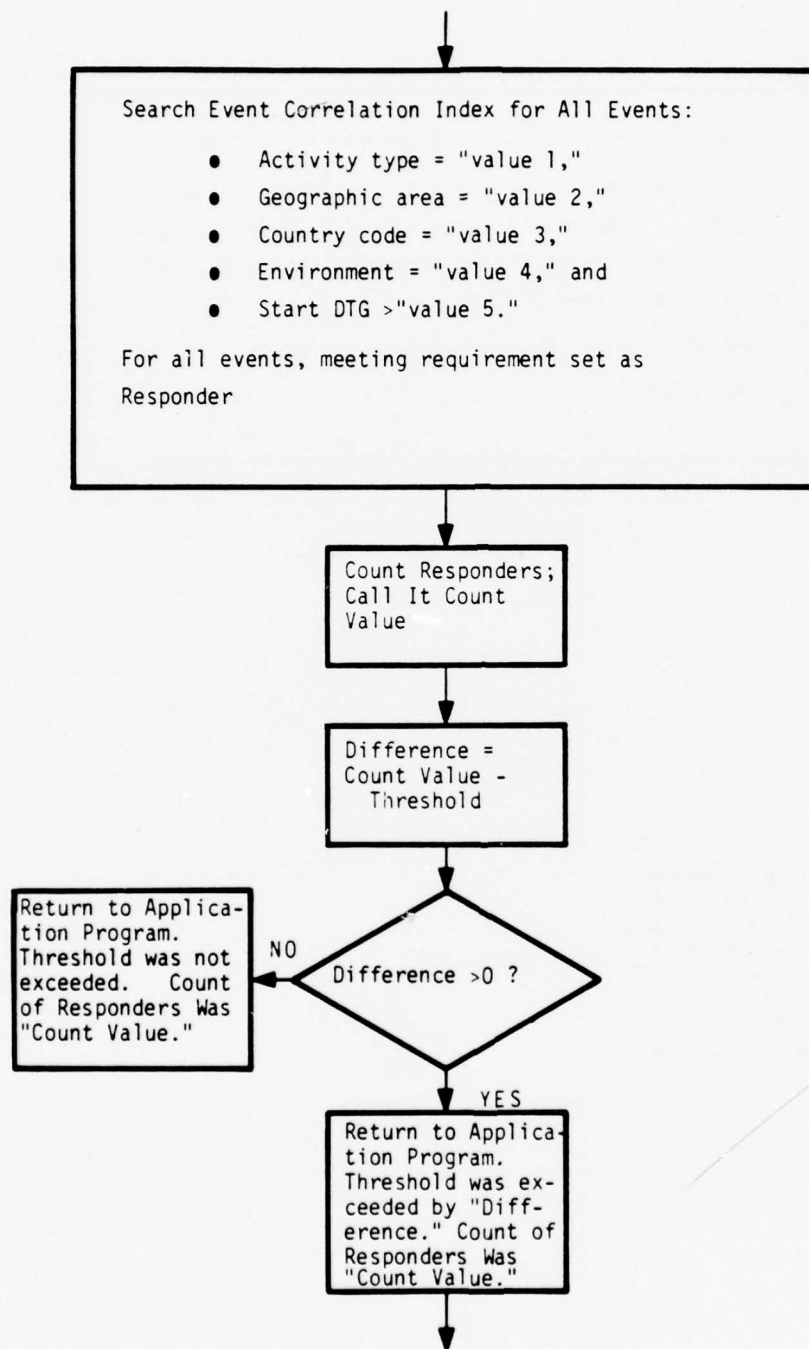


Figure 24. Second Sample Query

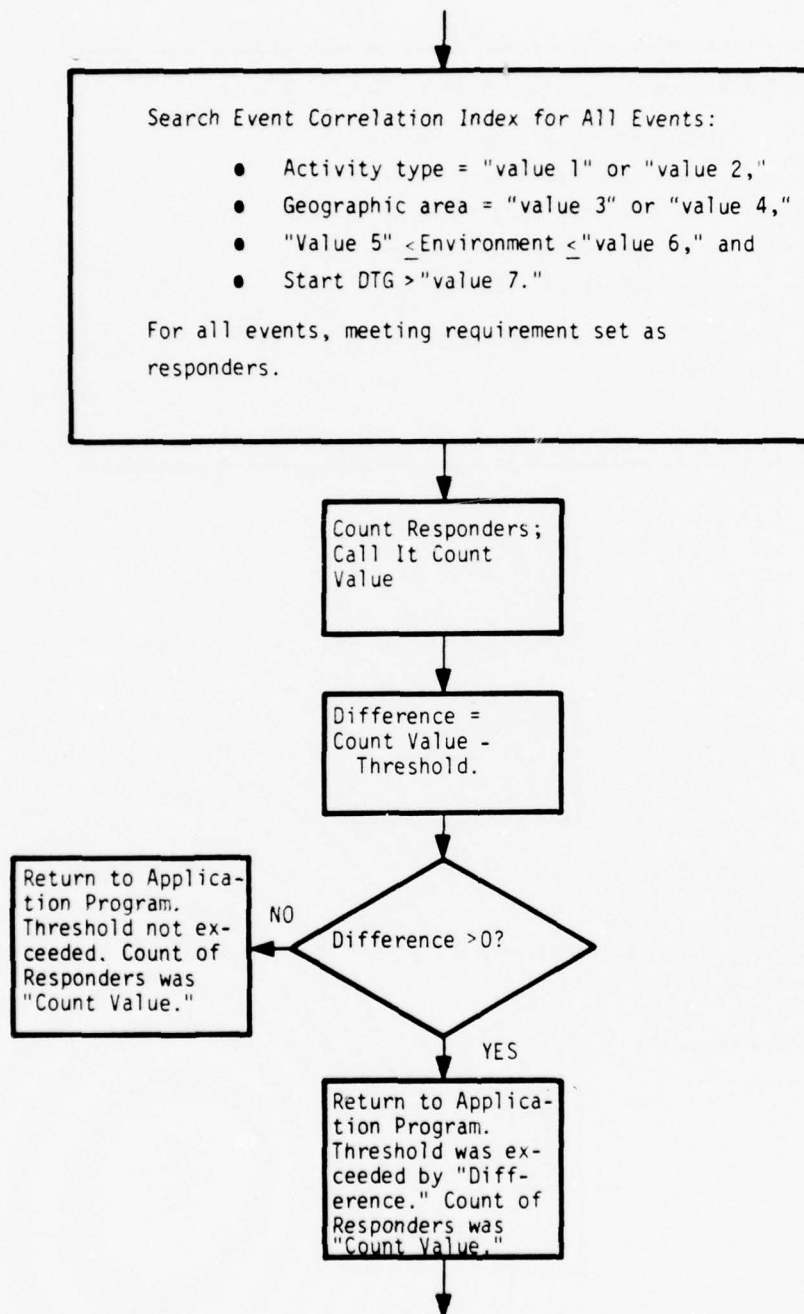


Figure 25. Third Sample Query

- Display the responders.
- Display the first "N" responders.
- Display the first "N" responders and be able to display another group of responders if there are any.
- Display the next group of "N" responders (a follow-on to the above).
- Increment a field in the responders (to keep an activity count).

Figures 26 through 36 show typical segments that could appear in a correlation query. Figures 26 and 27 illustrate possible first segments in a query (e.g., the search component). Many variations of the search component will exist providing for different combinations of search elements. Figures 28 through 36 show segments which can logically follow a search segment. Under some conditions, the segments can cascade on each other. Figures 37 through 40 show typical queries with various options for return to the analyst.

The correlation queries discussed above are simple with respect to time. They count, sum, and display occurrences in a single time interval. The term trend analysis gives rise to the concept of performing these queries over multiple time intervals. Thus, one could create a display to answer the following question:

What is the average number of events with activity type "value" or "value" or "value," in geographic area "value" or "value," and in an environment between "value" and "value" that have occurred per hour in each of the time periods: last year, last six months, last month, last week, last day, and last hour?

This query would result in a bar graph with an entry for each of the six defined time periods. Such a graph would show the activity trend as well as current activity. Figure 41 contains a functional flowchart for this query, and Figure 42 shows what a sample display might look like.

2.3.3 Workload

The dominant factors in system workload are the maintenance of the Message Index and Event Correlation Index and the performance of trend and correlation analysis. These functions are also time-sensitive and need to be performed in near-real-time.

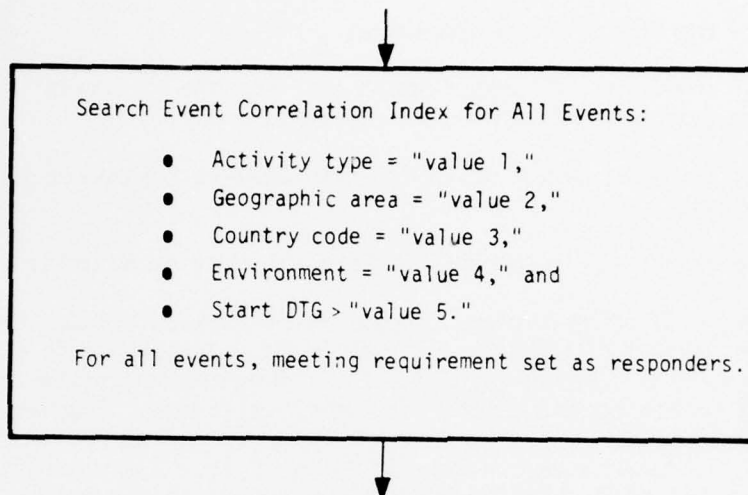


Figure 26. Simple Search Segment

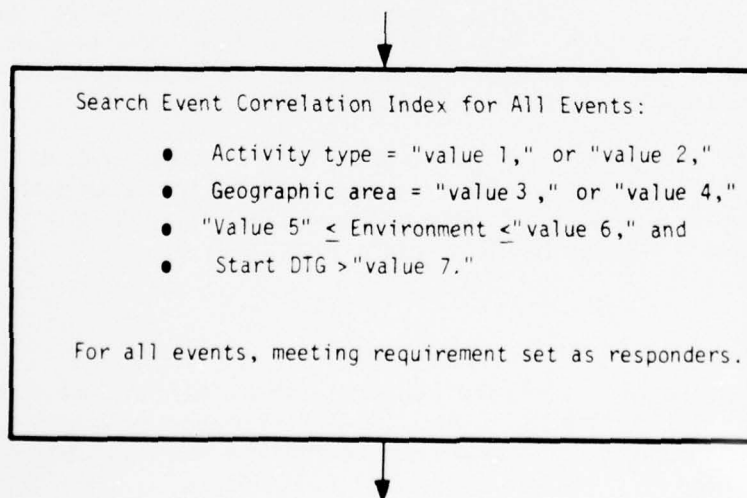


Figure 27. More-Complex Search Segment

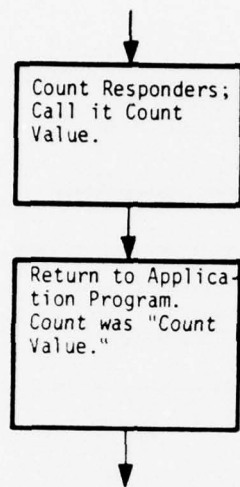


Figure 28. Count Responders Segment

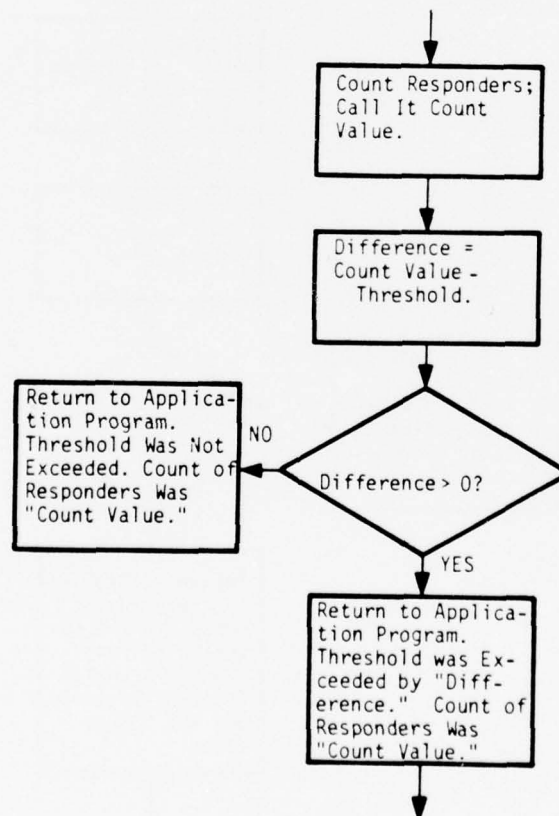


Figure 29. Count Responders Exceed Threshold Segment

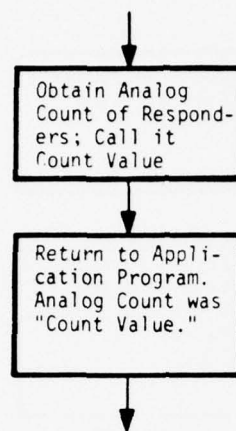


Figure 30. Analog Count Responders Segment

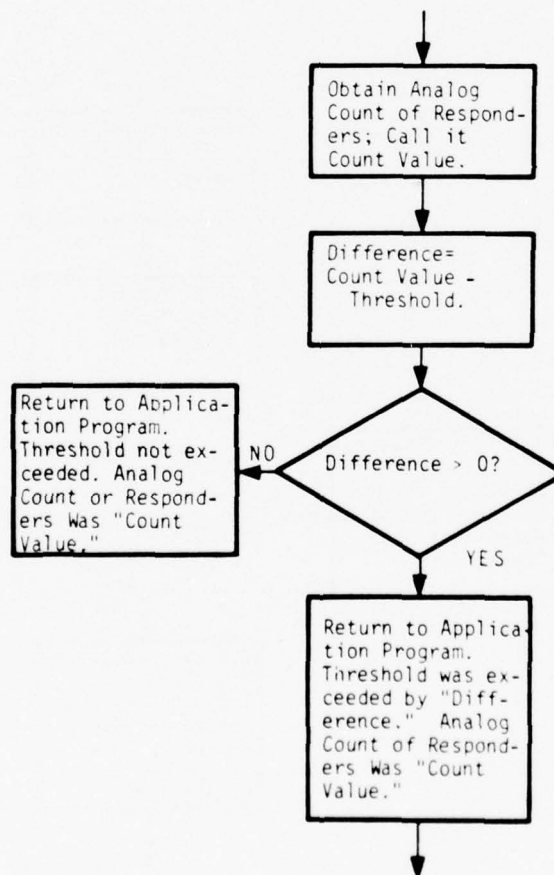


Figure 31. Analog Count Responders Exceed Threshold Segment

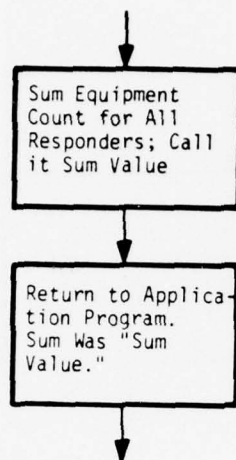


Figure 32. Sum Field Segment

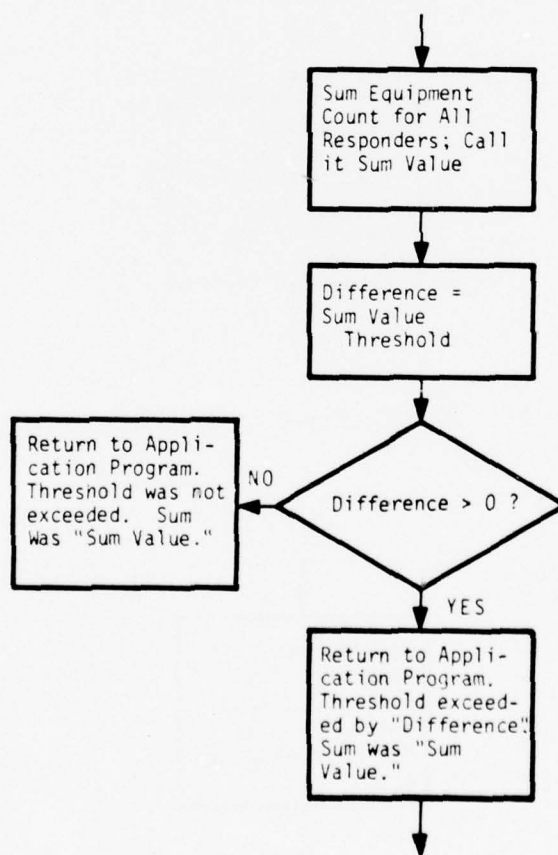


Figure 33. Sum Field Exceed Threshold Segment

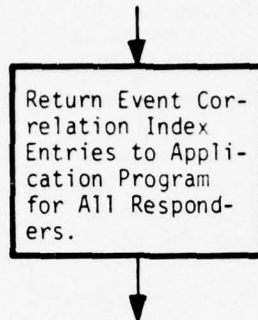


Figure 34. Return All Responders Segment

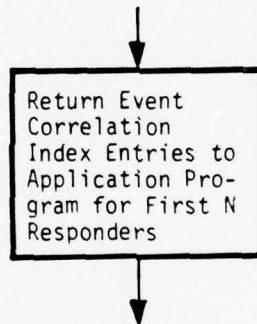


Figure 35. Return First N Responders Segment

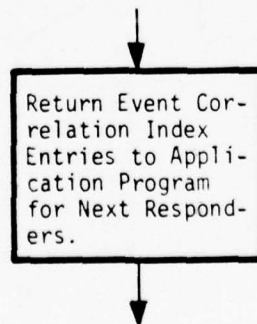


Figure 36. Return Next N Responders Segment

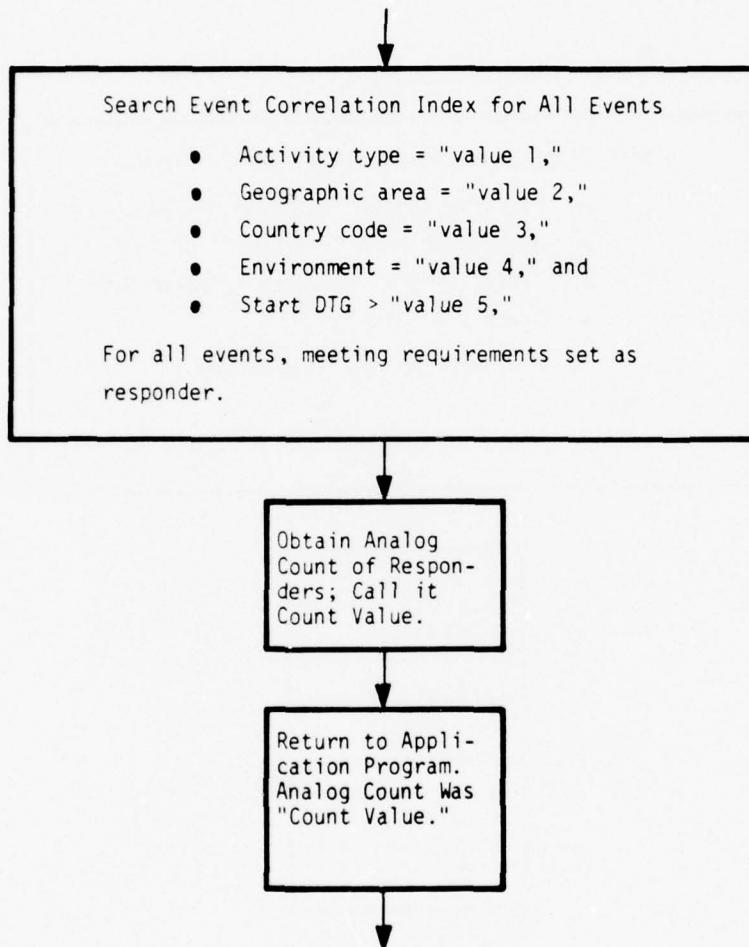


Figure 37. Typical Query Search

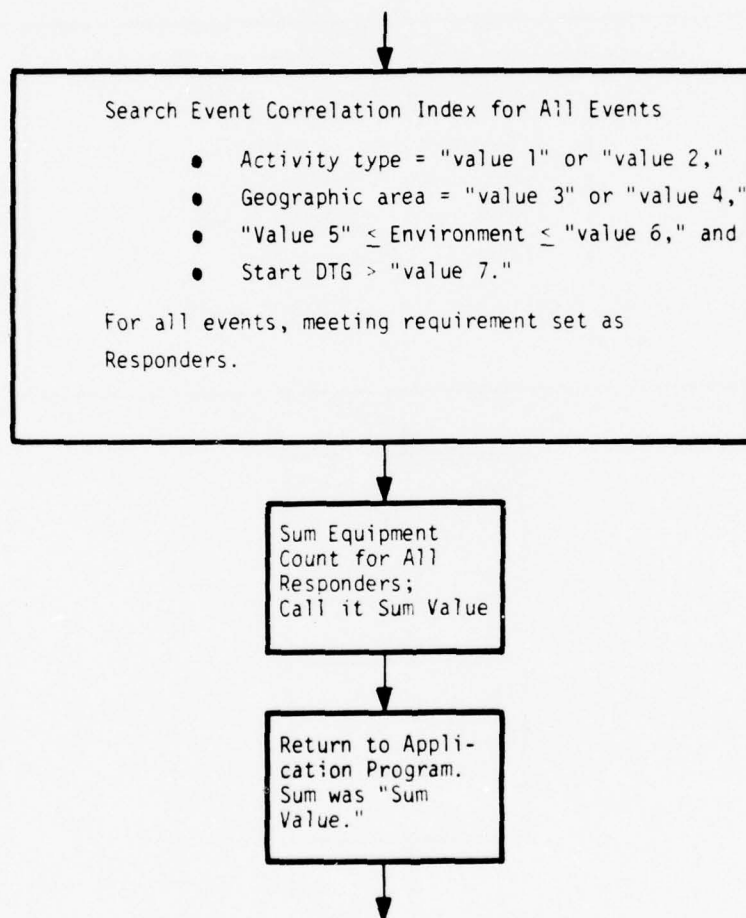


Figure 38. Typical Query with More Complexity

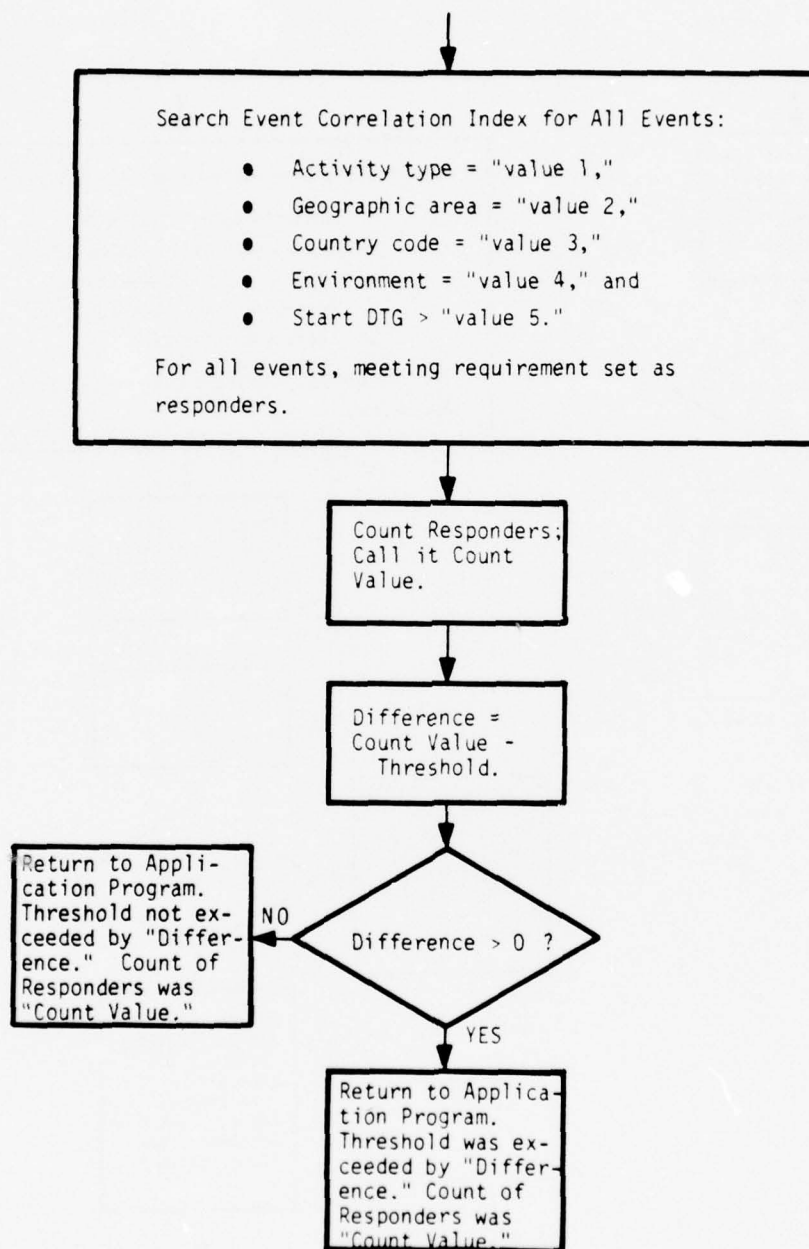


Figure 39. Typical Query with Threshold Check

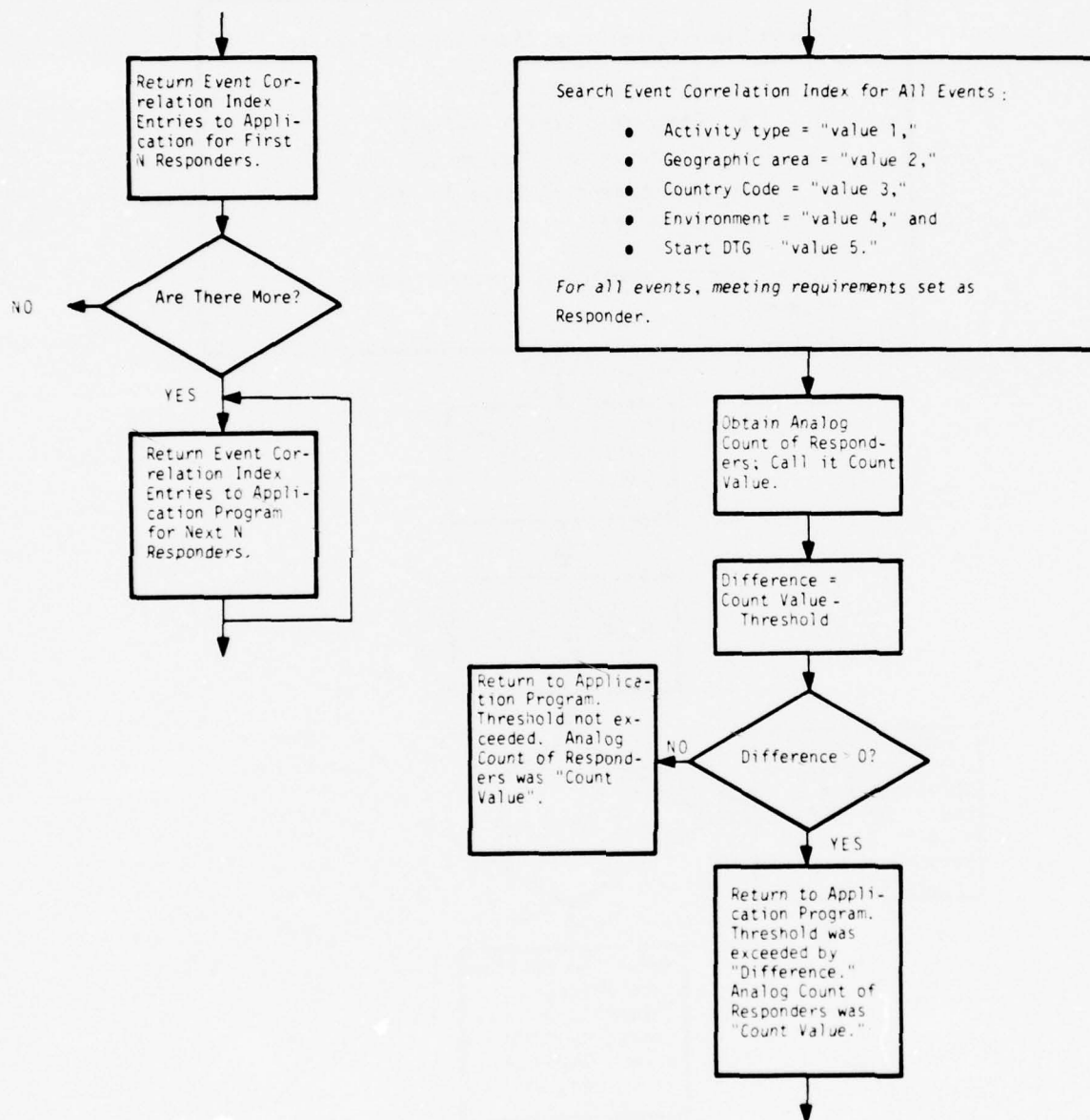


Figure 40. Typical Query with Return of Events of Threshold Exceeded

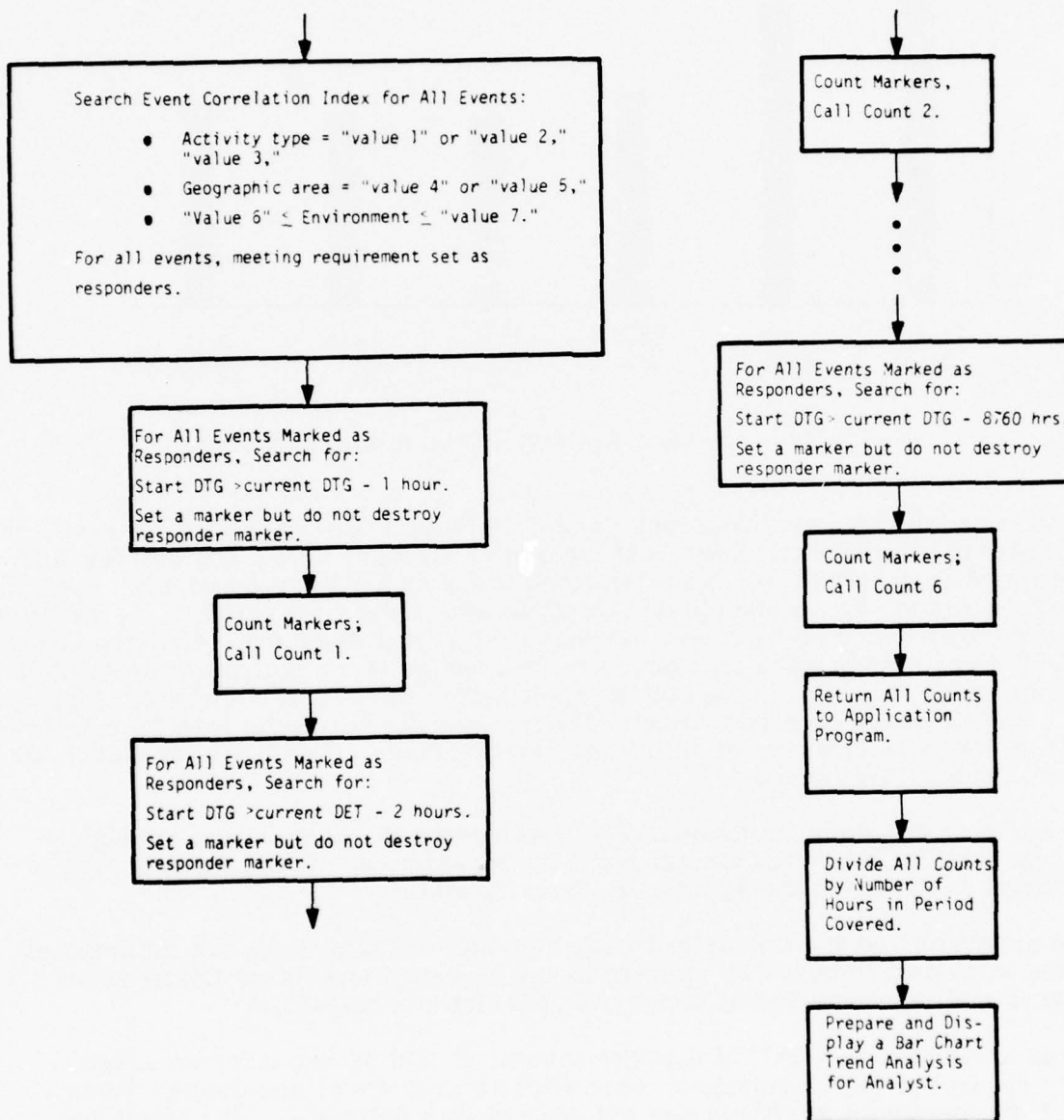


Figure 41. Procedure for Preparing a Chart of Activity Over Time

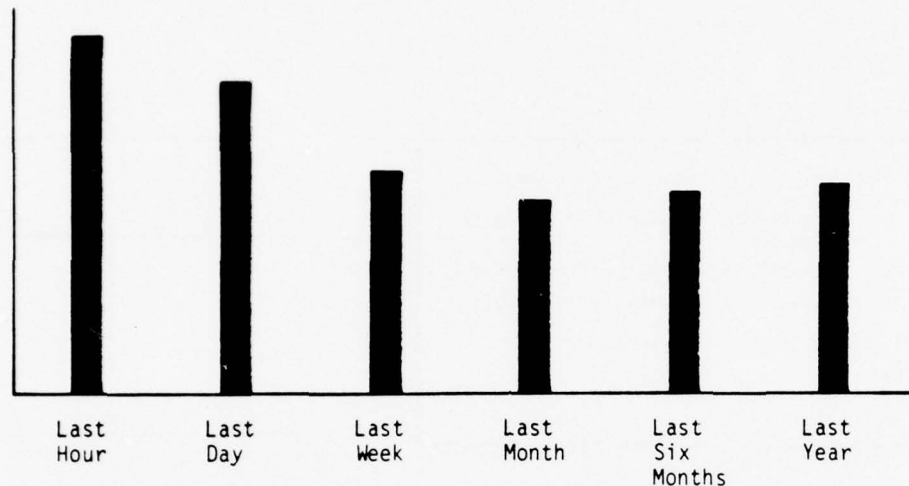


Figure 42. Average Activity in the Indicated Period

Updates (additions, modifications, and deletions) to the L/O/E Indexes are not time-sensitive, can wait a while if the system is very busy, and are few and infrequent in occurrence. New locations (such as air bases) and new types of equipment (such as airplanes) do not come into being very often. Organizational changes may occur more often, but they, too, are infrequent. Updates to the L/O/E Correlation Index will occur more frequently as equipment or organizations move, but this is, again, a small factor in system workload. The total workload imposed by maintenance of these indexes should be less than 1 percent of the estimation error in the workload associated with the Message and Event Correlation Indexes.

It could be necessary occasionally to reapportion the associative storage to increase or decrease the size of some of the indexes. This is a task that is performed only when the system is virtually idle.

It is apparent that the throughput requirements of the system are determined by the workload imposed by updates to the Message and Event Correlation Indexes and performance of trend and correlation analysis.

It was estimated in WPO234 that the system should be sized for an average daily receipt of 3000 messages, each with an average of one event. Subsequent discussion of workload has not altered this estimate. The trend and correlation analysis workload has, however, gone up substantially, with the current estimates being that about 13 searches per event will constitute the standard, automatically evoked sets. This estimate has gone up twice, and one should expect it to rise further as the capabilities of a system such as ECAM become available to the analysts. They are conceptually bound at

present by hardware constraints. As these are removed, they will be able to perform analyses not formerly considered. Thus, the workload from these sources is expected to be:

- An average of 3000 Message Index additions per day
- An average of 3000 Event Correlation Index additions per day
- An average of 3000 Message Index purges per day
- An average of 3000 Event Correlation Index purges per day
- An average of 39,000 standard correlation queries per day
- An estimate of an additional 4000 (approximately 10 percent more) ad hoc correlation queries per day

2.4 REQUIREMENTS SUMMARY

2.4.1 Data Base Storage Requirements

2.4.1.1 Data Base Size -- An estimate of the size of the SACWARDANS directories data base is shown in Table 9. This estimate, of approximately 37.6 million eight-bit bytes, is exclusive of overhead for format identification, work space, and unusable waste space. The percentage of additional space required for overhead is highly variable, being primarily a function of the array architecture and also a function of the manner in which the array is used. In previous analysis reported in WPO234, in excess of 30 percent additional space was required for overhead. Based on that analysis, the total storage space requirement is approximately 49 million bytes.

A substantially smaller associative store could be used for a test bed system. Thirty days of event and message entries plus all of the other indexes can be accommodated in a 10-million-byte store, with approximately 40 percent additional space being available for overhead. Thirty days of message/event traffic is a meaningful and useful data base. This size data base is sufficiently large to service a portion of the operational requirement. Such a system would provide for study of the current concept and would provide the means for investigation of additional analysis procedures.

2.4.1.2 Record Sizes -- As described in Section 2.2, the directories data base contains six types of records, of which two types predominate -- over 95 percent of the record instances are either Message Index entries or Event Correlation Index entries. All of the record types require 50 or fewer bytes per entry, exclusive of overhead. Table 10 lists the percentage distribution of record instance lengths.

Table 10. Distribution of Record Lengths

Number of Bytes (8 bits/byte)	Percent of Records With This Number of Bytes	Percent of Records With This Number or Fewer Bytes
13	2.7	2.7
26	48.2	50.9
27	0.1	51.0
36	4.0	51.4
42	48.2	99.6
50	0.4	100.0

2.4.1.3 Item Sizes -- Section 2.2 also describes the items which make up the directories data base records. Table 11 lists the percentage distribution of item instance lengths. These percentages include the number of instances of the records which contain the items.

Table 11. Distribution of Item Lengths

Number of Bytes (8 bits/byte)	Percent of Entries With This Byte Size	Percent of Entries With This or Fewer Bytes
1	14.0	14.0
2	14.4	28.4
3	~0	28.5
4	33.4	62.0
5	33.2	95.2
9	4.7	99.9
12	~0	99.9
14	~0	99.9
228	~0	100.0

2.4.2 Algorithms

Functional flowcharts are presented in Section 2.3 for the algorithms required to maintain the SACWARDANS directories data base as it was described in Section 2.2. Section 2.3 also contains typical queries, with the corresponding search algorithms. These algorithms also are shown in functional form; factors introduced by the manner of storing the directories are not shown.

Tables 12 through 26 contain a breakout of the ECAM-related operations for each of the functional algorithms shown in Section 2.3 that support data base maintenance (add, delete, and modify entries). These charts list each operation (transfer parameter, exact match search, etc.), showing the size of the operand involved. They also indicate the expected number of hits and other results data. These charts reflect single executions of the algorithms. Table 27 presents an estimate of ECAM operation usage in data base maintenance functions. Information concerning parameter lengths is also provided. These estimates were derived by applying the expected number of executions of each algorithm per day to the data contained in Tables 12 through 26. The frequency of execution of the algorithms is functionally dependent upon the expected update rate for the various indexes. Since several of these indexes have a low update rate, the frequency of execution of the corresponding algorithms is low, and hence they contribute little to the composite workload. The algorithms which dominate in the calculation of workload are those for Add Event, Add Message, and Purge Messages and Events. Some additional workload was factored in to account for surges in the activity of the infrequently used algorithms and to account for ad hoc requests not accounted for by the standard spectrum of data base maintenance activity.

Table 12. Use of Array in Algorithm for Add Equipment

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 4 parameters to ECAM.	1 of <15 3 of <5	
Search exact match.	~ 15	Test for no hits.
Search exact match.	~ 5	Test for no hits.
Search exact match.	~ 5	Test for exactly 1 hit.
Search exact match.	~ 5	Test for exactly 1 hit.
Search for empty entry.		Find first empty.
Write the 4 parameters into array as new entry and set nonempty.	1 of ~15 3 of ~5	

Table 13. Use of Array in Algorithm for Add Organization

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 3 parameters to ECAM.	1 of <30 1 of <5 p of <3	
Search exact match.	~30	Test for no hits.
Search exact match.	~5	Test for no hits.
Search exact match.	~3	Test for exactly 1 hit.
Search for empty entry.		Find first empty.
Write the 3 parameters into array as new entry and set nonempty.	1 of ~30 1 of ~5 1 of ~3	

Table 14. Use of Array in Algorithm for Add Location

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer of 6 parameters to ECAM.	1 of ~30 1 of ~12 4 of <4	
Search exact match.	~30	Test for no hits.
Search exact match.	~12	Test for no hits.
Search exact match.	<4	Test for no hits.
Search exact match.	<4	Test for exactly 1 hit.
Search exact match.	<4	Test for exactly 1 hit.
Search exact match.	<4	Test for exactly 1 hit.
Search for empty entry.		Find first empty.
Write the 6 parameters into array as new entry and set nonempty.	1 of ~30 1 of ~12 4 of <4	

Table 15. Use of Array in Algorithm for Add
Location/Organization/Equipment

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 4 parameters to ECAM.	3 of <5 1 of <2	
Search exact match.	~5	Test for exactly 1 hit.
Search exact match.	~5	Test for exactly 1 hit.
Search exact match.	~5	Test for exactly 1 hit.
Search for empty entry.		Find first empty.
Write the 4 parameters into array as new entry and set nonempty.	3 of ~5 1 of ~2	

Table 16. Use of Array in Algorithm for Add Message

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 6 parameters to ECAM.	1 of 9 2 of 5 3 of <3	
Search exact match.	~3	Test for no hits.
Search exact match.	9	Test for exactly 1 hit.
Write the 6 parameters into array as new entry and set nonempty.		

Table 17. Use of Array in Algorithm for Add Event

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 15 parameters to ECAM.	2 of 5 3 of ~5 5 of ~3 2 of ~2 3 of ~1	
Search exact match.	~3	Test for no hits.
Search exact match.	~3	Test for no hits.
Search exact match.	~3	Test for no hits.
Search exact match.	~3	Test for no hits.
Search exact match.	~5	Test for no hits.
Search exact match.	~5	Test for no hits.
Search exact match.	~5	Test for no hits.
Search exact match.	~1	Test for no hits.
Search exact match.	~1	Test for no hits.
Search exact match.	~2	Test for no hits.
Search exact match.	~1	Test for no hits.
Search exact match, and	~3	
Search exact match, and	~3	
Search exact match, and	~5	
Search exact match, and	~5	
Search exact match, and	~1	
Search exact match, and	~5	
Search between limits, and	5	
Search between limits.	5	Test for no hits; if there are any, pass back all 15 parameters of events meeting criteria.
Search for empty entry.		Find first empty.
Write the 15 parameters into array as new entry and set nonempty.		

Table 18. Use of Array in Algorithm for Modify Equipment

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	~5	Test for exactly 1 hit.
Search exact match.	~5	
Return entry to host. Transfer 4 parameters.	~27	
Perform Delete Equipment.		
Perform Add Equipment.		

Table 19. Use of Array in Algorithm for Modify Organization

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	~5	Test for exactly 1 hit.
Search exact match.	~5	
Return entry to host. Transfer 3 parameters.	~36	
Perform Delete Organization.		
Perform Add Organization.		

Table 20. Use of Array in Algorithm for Modify Location

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	~3	Test for exactly 1 hit.
Search exact match.	~3	
Return entry to host. Transfer 6 parameters.	~50	
Perform Delete Location.		
Perform Add Location.		

Table 21. Use of Array in Algorithm for Modify
Location/Organization/Equipment

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 3 parameters to ECAM.	2 of ~5 1 of ~3	Test for exactly 1 hit.
Search exact match, and	~5	
Search exact match, and	~5	
Search exact match.	~3	
Return entry to host. Transfer 4 parameters.		
Perform Delete L/O/E.		
Perform Add L/O/E.		

Table 22. Use of Array in Algorithm for Delete Equipment

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	~5	Test for exactly 1 hit.
Search exact match.	~5	
Write zeros into entry.	27	
Mark format field with NOT IN USE.		

Table 23. Use of Array in Algorithm for Delete Organization

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	~5	Test for exactly 1 hit.
Search exact match.	~5	
Write zeros into entry.	~36	
Mark format field with NOT IN USE.		

Table 24. Use of Array in Algorithm for Delete Location

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	~3	Test for exactly 1 hit.
Search exact match.	~3	
Write zeros into entry.	~50	
Mark format field with NOT IN USE.		

Table 25. Use of Array in Algorithm for Delete Location/Organization/Equipment

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 3 parameters.	2 of ~5 1 of ~3	Test for exactly 1 hit.
Search exact match, and	~3	
Search exact match, and	~5	
Search exact match.	~5	
Write zeros into entry.	13	
Mark format field with NOT IN USE.		

Table 26. Use of Array in Algorithm for Purge Messages and Events

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 1 parameter to ECAM.	5	Should have ~3000 hits if run one each day.
Search, less than	5	
Read parameter from first (next) marked entry.		
Search exact match, or	~3	Should have ~3000 hits total per day. Each instance of search would average 1 hit.
Search exact match.	~3	Should have ~3000 hits total per day. Each instance of search would average 1 hit.
Write zeros into entry.	~42	Total of 3000 entries per day.
Write zeros into entry.	~26	Total of 3000 entries per day.
Mark format field with NOT IN USE.		Total of 6000 entries per day.

Table 27. Estimated Use of Array Operations for Data Base Management (Updates, Additions, and Deletions)

Operation	Usage
a) <u>Searches</u>	
<u>Total:</u>	90,000
<u>Single-field:</u>	50,000
Exact match	50,000
Less than	Negligible
Greater than	Negligible
Between limits	Negligible
<u>Multiple-field:</u>	6,000
Add used	21,000
Or used	3,000
Eight fields	3,000
Two fields	3,000
Exact match	24,000
Between limits	6,000
<u>Parameter:</u>	
Less than or equal to 5 bytes	>95%
Less than or equal to 10 bytes	>98%
Longest parameter	≤30 bytes
b) <u>Parameter Transfers to/from ECAM</u>	
<u>Total:</u>	100,000
To ECAM	65,000
From ECAM	20,000
<u>Parameter lengths:</u>	
Less than or equal to 5 bytes	>95%
Less than or equal to 10 bytes	>98%
Longest parameter	≤30 bytes

Table 27. Estimated Use of Array Operations for Data Base Management
(Updates, Additions, and Deletions) (Concluded)

Operation	Usage
c) <u>Write Parameter into Field</u>	
<u>Total:</u>	80,000
<u>Parameter length:</u>	
Less than or equal to 5 bytes	95%
Less than or equal to 10 bytes	98%
Longest parameter	≤30 bytes
d) <u>Read Parameter from Field</u>	
<u>Total:</u>	25,000
<u>Parameter length:</u>	
Less than or equal to 5 bytes	98%
Less than or equal to 10 bytes	99%
Longest parameter	≤30 bytes
e) <u>Zero Entry</u>	
<u>Total:</u>	7,000
<u>Entry length:</u>	
Longest	50 bytes
f) <u>Examine for Number of Responders</u>	
<u>Total:</u>	50,000
<u>Expected number:</u>	
Zero	40,000
One	4,000
One to 10 range	3,000
Three thousand	1
g) <u>Find First Responder</u>	7,000

Data base maintenance contributes fractionally to the ECAM workload. The dominate factor is the execution of queries -- both standard and ad hoc. Standard queries are standard only in the sense that once defined they are automatically evoked under a specified set of circumstances. The query sequence to be applied in a given circumstance and the circumstances in which a sequence is to be applied may both be redefined as required by the analysts. Thus, a "standard query sequence" is a sequence which is currently defined for automatic evocation. These definitions will, of course, change. What is not likely to change is the nature of the query algorithms. They basically consist of counting the number of occurrences during a given time interval meeting a set of criteria. The information required to qualify occurrences (events) is contained in the Event Correlation Index entries. Hence, selection of events to be counted as responding to a query criteria is determined by a sequence of searches, with the results being logically combined using "and" and "or" operations. Ad hoc queries, against the Event Correlation Index, are similar in nature.

The workload derivable from query operations is not well defined -- either in terms of the quantity of queries or in terms of the complexity of the queries. This stems from a lack of analyst experience in an environment which can perform the extensive -- both in quantity and complexity -- queries possible with an associative system. Given a specific query, it is easy to show the search requirements for the query. Tables 28 through 30 present breakdowns for the three sample queries shown in Section 2.2, Figures 23 through 25.

Table 28. Use of Array in Algorithm for First Sample Query

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 5 parameters to ECAM.	1 of 5 1 of ~2 3 of ~1	Multiple responders likely.
Search exact match, and	~1	
Search exact match, and	~1	
Search exact match, and	~2	
Search exact match, and	~1	
Search greater than.	~5	
Count responders.		
Return count to host.	2 of 3	

Table 29. Use of Array in Algorithm for Second Sample Query

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 6 parameters to ECAM.	1 of 5 1 of ~2 3 of ~1 1 of ~3	Multiple responders likely.
Search exact match, and	~1	
Search exact match, and	~1	
Search exact match, and	~2	
Search exact match, and	~1	
Search greater than.	~5	Only if threshold was exceeded.
Count responders.		
Return count to host.	2 of 3	
Return difference to host.	2 of 3	

Table 30. Use of Array in Algorithm for Third Sample Query

Operation Type	Operand Size, Bytes (8 bits/byte)	Result
Transfer 8 parameters to ECAM.	1 of 5 5 of ~1 2 of 2 or 3	Multiple responders likely.
Search exact match, or	~1	
Search exact match, and	~1	
Search exact match, or	~1	
Search exact match, and	~1	
Search between limits, and	~1	Only if threshold was exceeded.
Search greater than.	5	
Count responders.		
Return count to host.	2 or 3	
Return difference to host.	2 or 3	

The estimate for the number of new events per day is 3000. It is also estimated that each event will trigger a standard sequence of approximately 13 queries. This results in an estimate of 39,000 standard queries to be executed per day. Analysts will be expected to initiate other ad hoc queries. An additional query workload of 4000 queries (approximately 10 percent additional) is allowed in our figures for ad hoc queries. Thus, the query workload is 43,000 queries per day.

As indicated above, the confidence level in the query workload estimate is low. Further, the workload could increase substantially during a period of crisis. In a crisis environment, the number of events is sure to increase. The number of queries resulting in the triggering of additional processing will also increase. Determining the nature of the unusual occurrences is likely to result in additional processing (additional queries) and readback to the host of additional information concerning the events. It seems clear that the system should have processing capacity considerably in excess of estimates of average workload in order to accommodate surges resulting from higher short-term, real-world activity levels. Hence, we have further extrapolated workload estimates to account for an additional 50 percent query workload.

Table 31 contains workload information assuming 43,000 and 65,000 queries per day and assuming an equal proportion of the three sample queries.

Table 31. Query Workload Assuming Sample Queries

Operation	43,000 Queries/Day	65,000 Queries/Day
Parameters to transfer to ECAM	~273,000	~412,000
Bytes of parameters	~600,000	~900,000
Searches	~230,000	~350,000
Exact match searches	~172,000	~260,000
Between limit searches	~15,000	~25,000
Greater than searches	~43,000	~65,000
Results to be returned to host	~72,000	~110,000
Bytes to be returned to host	~215,000	~325,000

Analysis of the Event Correlation Index content shows that there are 11 fields which could meaningfully serve as search fields in tallying the number of occurrences meeting a set of criteria. Of these, two fields are DTG fields, and hence they would frequently involve searches other than exact match. The other nine would normally be searched for exact match, but searches involving OR (e.g., equipment type equal A or B) are possible. Table 32 shows the equivalent of the data in Table 31 under the assumption that all 11 fields are searched in each query, with nine fields being exact match searched against a single argument. This represents an extreme workload. A more reasonable high level of workload is shown in Table 33. It assumes that seven fields are searched, with eight arguments being transferred to the ECAM. These searches may result from search of seven different fields or search of the same field two or more times. Clearly, some queries might require only four field searches, while others might require nine or 10. The number seven is an average which is on the high side compared with the current statement of the SACWARDANS query requirements. This table also assumes a high number of queries (85,000), which is double the currently stated estimate and much higher than earlier estimates.

Table 32. Query Workload Assuming Extreme of 11 Fields Searched per Query

Operation	43,000 Queries/Day	65,000 Queries/Day
Parameters to transfer to ECAM	~559,000	~845,000
Bytes of parameters	~1,978,000	~2,990,000
Searches	~473,000	~715,000
Exact match searches	~387,000	~585,000
Other searches	~86,000	~130,000
Results to be returned to host	~86,000	~130,000
Bytes to be returned to host	~260,000	~390,000

Table 33. Query Workload -- High Expected Value

Operation	85,000 Queries/Day
Parameters to transfer to ECAM	~680,000
Bytes of parameters	~2,125,000
Searches	~595,000
Results to be returned to host	~170,000
Bytes to be returned to host	~510,000

Tables 32 and 33 contain estimates of return parameters based on return of two values: the count of events and the difference between this value and a threshold. It is assumed that this averages out to six bytes per query.

Additional workload occurs if event IDs are sometimes returned. Analysts might in some cases require the message IDs associated with an event rather than, or in addition to, the event ID. The other values in the Event Index records can be obtained easily from disk copies, given the ID. Hence, return of these values is not a requirement. It may be the simplest way to obtain the values, and if the system is not busy, the channels could undoubtedly support the return of entire Event Index entries. It is not necessary to design the system with this capacity during periods of heavy workload.

Analysts will not be able to inspect large numbers of event or message entries. If one assumes that there are 15 analysts working on a round-the-clock basis, then there are 21,600 analyst minutes per day. It is hard to imagine them scanning 10 entries per minute. This would generate a requirement for 216,000 IDs to be transferred. Since they might terminate some scans before they were all examined, this could require as many as 500,000 IDs to be transferred. At five bytes (for messages) per ID, this is 2.5 million bytes, which is less than 30 bytes per second.

Table 34 adds a factor for the return of IDs to the estimates in Table 33.

Table 34. Query Workload -- High Expected Value,
Including ID Return

Operation	85,000 Queries/Day
Parameters to transfer to ECAM	~680,000
Bytes of parameters	~2,125,000
Searches	~595,000
Results to be returned to host	~700,000
Bytes to be returned to host	~3,100,000

As can be seen from Section 2.3 and the preceding discussion, both the data base management and query algorithms are basically simple. The quantities are high, especially when related to conventional data base management and query access. But the complexity is not great. The algorithms make extensive use of the exact match search, generally with short arguments. Other searches are used, but at a much lower frequency. The other frequently used operations are those which count the number of responders and sum a field within all records satisfying the search criteria. The algorithms are structurally simple, containing successions of searches which may be logically

AD-A037 834

HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
ADVANCED LOGIC TECHNOLOGY.(U)

FEB 77 G A ANDERSON, E D JENSEN, R Y KAIN

F30602-75-C-0148

UNCLASSIFIED

F0375-FR

RADC-TR-76-388

NL

2 of 4
ADA037834



ANDed and ORed together. Generally, the searches can be ordered in a manner convenient to the system. They must, of course, adhere to the proper logic, but the constraints are minor in terms of search order. The algorithms do not, with the exception of Purge Messages and Events, contain high-level language loops. The algorithms generally can be performed as an entity without interrupts for user intervention. Where interaction with the user is required, then the search results could be saved within the array. However, a simpler procedure which meets the requirement is to interrupt for user intervention and then restart the algorithm from the start, thus eliminating the problems which will arise from retaining intermediate results and restarting from that point. Exception processing can be similarly treated.

2.4.3 Use Factors

SACWARDANS will initially have seven analysts on-line at a time. This is likely to expand to 12 to 15 analysts.

Initially, SACWARDANS was planned for implementation on a single host 6080. This inferred that the ECAM would interface to a single computer, with another computer as backup. This meets the reliability requirement. However, system flexibility is enhanced by a dual interface, with either 6080 having access at any time. Thus, while not a requirement, this dual interface is a desirable feature.

SACWARDANS is an on-line system. Analysts sitting at terminals require a reasonable response time. It is generally thought that response time in excess of 1 to 2 seconds is not acceptable to users; they become restless and lose their train of thought.

SECTION 3

SOFTWARE STRUCTURE

Although the major emphasis in this initial contract was on refining the hardware design to show feasibility and to allow development of a plan, some initial software work was done to ensure that the resulting hardware would "fit" a realistic environment. The major processing functions that were identified and their assignment to hardware elements are shown in Figure 43. Each one is briefly discussed below.

3.1 HOST FUNCTIONS

All host functions are intended to run under an existing operating system (OS) and do not require that modifications to the OS be made. Host functions are:

- Query Processor -- This module receives query requests from users at terminals and from other jobs within the host system. It analyzes the queries to identify those portions which are to be executed in the ECAM, parsing them and generating an intermediate-form query text for transmission to the ECAM via the ECAM handler module.
- ECAM Handler -- This module operates as the front-end processor for the ECAM, batching requests, managing buffers, monitoring status, and serving as the interface to the host's I/O subsystem. It should be a general-purpose element designed to give extensive support and protection to the query-generating software and users.
- Fast I/O -- In contrast to the ECAM handler, this module is to be very specific to the task of ECAM checkpoint/restart operations. If the fast I/O path is made directly from the ECAM to the host's memory, this module will be responsible for providing buffer space within its own user-job storage areas. It then will pass buffer pointers to the ECAM using normal I/O facilities. Note that this approach requires that the fast I/O handler be unswappable. User jobs can easily be made unswappable under HIS-6080-GCOS and other operating systems.
- ECAM Device Driver -- This is the interface component of the I/O subsystem that is written specifically for communicating with the ECAM as a peripheral device.

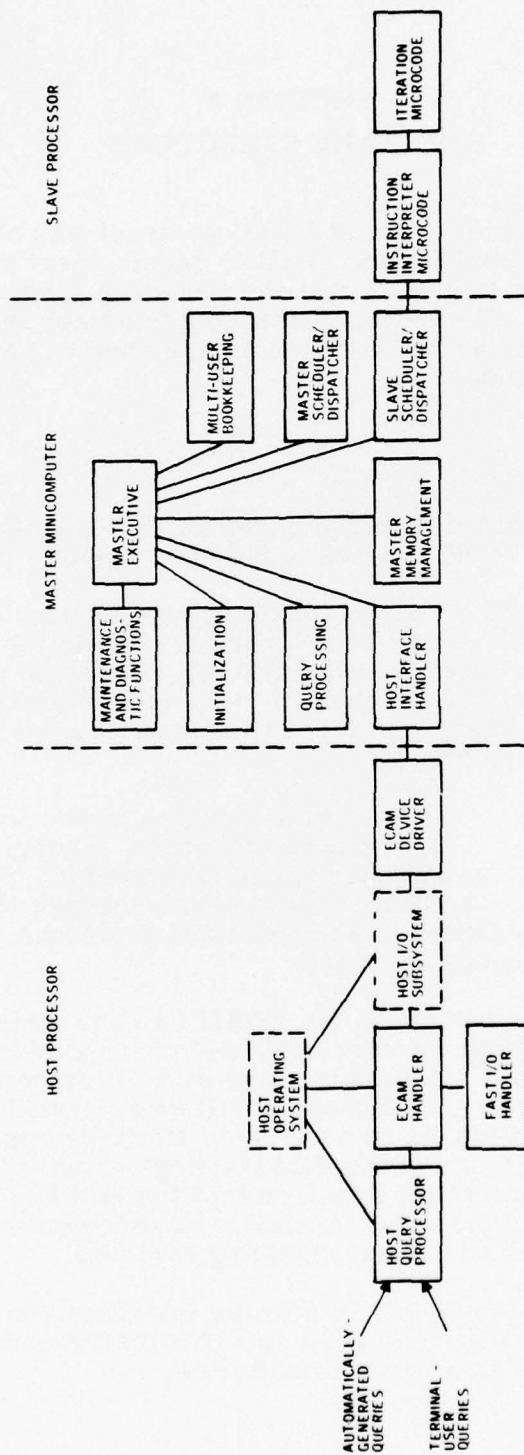


Figure 43. ECAM System Software Major Functions

3.2 MASTER MINICOMPUTER FUNCTIONS

The programs for the master are the major system software development items in the ECAM. For this reason, the contractor recommends use of a minicomputer (and its support software) instead of a more specialized unit. Master minicomputer functions are:

- Master Executive -- It is expected that some type of rudimentary executive will be developed to sequence among the master functions. It will be interrupt-driven, with the host interface and the interpreter as the major sources. For maintenance and diagnostic functions, it may be desirable to use portions of the minicomputer vendor's executive to handle peripherals.
- Maintenance and Diagnostic Functions -- These will include system integrity tests (*go/no-go*), background array testing tasks, and fault isolation routines.
- Initialization -- This module will handle creation of the descriptor table structure which defines the mapping of the data base logical storage scheme onto the physical storage of the array. The function may operate exclusively within the master or may be divided between the master and host. In either case, it includes the capability of working in a conversational mode with a data base administrator at a terminal and analyzing the packing and efficiency effects of alternative mappings.
- Query Processing -- It is expected that very little transformation of the queries will occur in the master. Instead, this module will be similar to a relocating loader, converting address references, adding user identification tags, etc.
- Host Interface Handler -- This module is concerned with the details of interface control and transferring data between the host I/O multiplexer and buffers in the master's memory.
- Master Memory Management -- There are several distinct memory management functions that must be performed in the master. It is expected that the most cost-effective approach to this function will not be to emphasize efficiency, however. Instead, the use of simple algorithms together with ample amounts of storage will probably result in the lowest total cost. The areas to be managed include:
 - Host communication buffers
 - Slave communication buffers
 - Slave working storage (in master memory)

- ECAM intermediate results (in array storage)
- Master working storage
- Multi-User Bookkeeping -- A considerable amount of master processing will be allocated to managing the pseudo-multi-programming of the ECAM required by the large number of users. This function must be in the master rather than the host; otherwise, multi-hosting is difficult. The functions to be performed include obtaining and releasing temporary storage for user activation records in master memory and for user intermediate results in the array, monitoring ECAM files to detect erroneous locks, and deadlock detection/prevention.
- Master Scheduler/Dispatcher -- This is the master program module that is concerned with sequencing the modules executing in the master central processor.
- Slave Scheduler/Dispatcher -- This module is similar to the one for the master except that slave preemption is not allowed, so state-saving and restoration requirements are simpler.

3.3 SLAVE PROCESSOR

Only two software-type functions are performed in the slave, and both are implemented as microprograms. The only software implications of these will be the requirement for an assembler for the interpreter.

SECTION 4

ECAM HARDWARE STRUCTURE

In this section, each of the functional elements of the ECAM is described. Prior to this, however, it is appropriate to briefly discuss the design philosophy and tradeoff criteria that were used and to justify the level of detail to which the design was taken.

It was known from the inception of the project that the performance of the ECAM was substantially in excess of that required for the baseline application. This characteristic caused system cost reduction, rather than speed, to be the major design goal.

In the array, implementation decisions did not greatly impact the software component of system costs, so design emphasis was on minimizing hardware costs. Because of its size, the array is also the biggest contributor to hardware cost. The cost minimization effort resulted in the following array characteristics:

- The design is very storage-technology independent. Because storage technology is advancing rapidly, both the first and subsequent systems should be able to take advantage of developments without significant software or hardware changes.
- Logic speeds are conservatively specified. The baseline shift rate is 1 microsecond per bit. Signal distribution for a large system running at this rate is straightforward and "unexotic."
- The number of wires in the signal distribution system is minimized by the choices of processing algorithms and by multiplexing. The cost of the distribution system is obviously a function of the number of wires. Less obviously, it has been found that the signal distribution system is also the biggest reliability problem in parallel machines.
- The word logic functions and the processing algorithms are chosen to minimize the size and pin count of the word logic integrated circuit.

In the control unit area, the situation was reversed in that system cost was best reduced by using hardware to reduce software cost. During the program, the approaches taken by many parallel machine designers were examined and, where possible, user experience data were reviewed. The results of work in this area are as follows:

- The data-handling and "host relations" functions are divorced from the function of controlling the sequencing of array operations. The master processor handles the former, the slave the latter.
- The use of a conventional minicomputer as the master is recommended in order to capitalize on existing software and programmer experience.
- The language interface to the ECAM is raised from the conventional sequence-of-primitives level to an HOL*-like intermediate language that can be easily generated from a compiler or written directly by a programmer.
- The mapping between the logical data storage schema and physical storage is specified by the user, but the query transformations that are necessary to reflect the mapping are handled by the hardware.
- The slave controller is designed to be changed as experience with the system results in ideas for improvement. Strategies are not hardwired.

In the subsections that follow, the hardware of the machine is described. Then, in Section 5, we describe the transformation of this hardware, via microcode, into a data base machine for SACWARDANS-type environments.

The level of detail developed for each of the hardware blocks has been chosen by the following criteria:

- Areas critical to performance (e. g. , the slave controller) and areas critical to cost and "producibility" (e. g. , the word logic and signal distribution) have been carried past the register level, to the point where detailed flow charts are provided and some generic logic has been specified.
- Areas specific to a host machine have been specified only at a functional level. Design of these is straightforward once a final choice is made.
- The design has been left relatively independent of the type of minicomputer chosen. Our work has not yet required detailed specification of interfaces to the master, so we have deferred a choice until software requirements are available.

*Higher-Order Language

The discussions below are separated into two subsections, covering first the control units and then the array.

4.1 CONTROL UNIT

As shown in Figure 1, the control unit consists of the host interface, the master control minicomputer with its memory, and the slave controller. The connection of these subunits is shown in Figure 44. These are described individually in the following subsections.

4.1.1 Master Control Processor

As was mentioned previously, our design work has not yet required that we commit to a particular minicomputer as the master processor. Our requirements for the machine are as follows:

- The minicomputer must allow for direct access to its main memory bus by the host interface and the slave controller. Access priority should be a round-robin or similar strategy to guarantee the host interface adequate bandwidth for block transfers.
- The minicomputer must have an interrupt capability sufficient to allow the slave and host interface to communicate errors. We expect that further definition of the host interface will produce additional requirements for the master's interrupt structure.
- The minicomputer must provide a programmed I/O facility by which the host interface and the slave can be controlled.
- The minicomputer's operating system must provide a static addressing environment so that the table structure and stacks used by the slave can be implemented as described. Dynamic relocation of data used by the slave is not allowed, nor are dynamic changes to any address-mapping mechanism.

The major source for speed requirements on the master will be the simulation efforts which occur early in the next phase. During this period, increased definition of the software will allow estimates of required memory sizes.

We expect that the major selection criterion on the minicomputer will be the cost of initial and maintenance programming. Minimization of these costs requires use of higher-order languages, availability of good standard software, and a high degree of programmer familiarity with the target machine. Based on this, we recommend that the PDP-11/45 be used as the baseline processor, but that a final decision be deferred until the software cost aspects of the ECAM application can be used in the selection process.

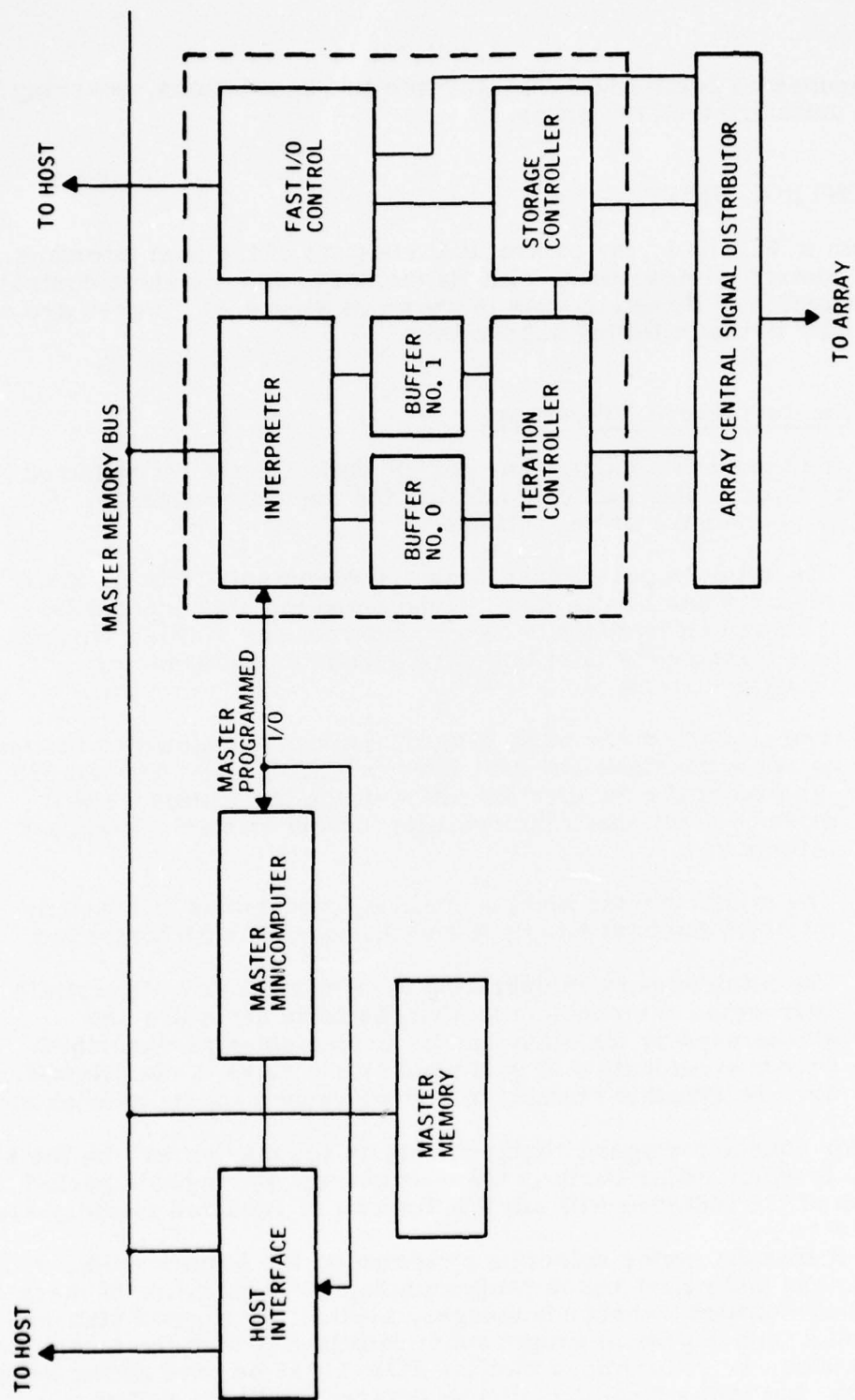


Figure 44. Control Unit Block Diagram

4.1.2 Slave Controller

4.1.2.1 Interpreter -- The Interpreter is a dedicated function computer; because the detailed algorithms of the interpreter function will evolve as experience is gained using the ECAM, the Interpreter is highly flexible and modular. The Interpreter consists of an application-independent Kernel Machine (KM), and ECAM-dependent Functional Modules (FMs). The KM is described next in Section 4.1.2.1.1, after which the FMs are described in Section 4.1.2.1.2. The KM provides flexible operand addressing mechanisms, while the actual operand storage units (register files, master interface registers, etc.) needed for efficient execution of the initial ECAM algorithms are FMs, as described in Section 4.1.2.1.2.1. Similarly, the KM includes the mechanism for invoking operations, while the actual operators which perform the required ECAM operations (such as Add, Shift, etc.) are FMs, as described in Section 4.1.2.1.2.2.

The KM consists of a Fetch Unit (FU) and a set of control and data lines to which the FMs are attached, as shown in Figure 45. The FU is vertically microprogrammed, having highly capable microinstruction sequencing logic which supports structured microcode, as described later.

4.1.2.1.1 Kernel Machine -- Physically, the Kernel Machine consists of a Fetch Unit and a set of control and data lines to which the Functional Modules are attached. The FMs are operators (such as adders, shifters, etc.) and storage (registers and memories); they are selected for efficient execution of the algorithms of a specific application. The use of different FMs, and changes in FM designs, do not affect the KM hardware. The FU is driven by microcode stored in a microprogram memory; its interface to the microprogram memory is asynchronous, so size and speed specification has been deferred until simulation results are available.

4.1.2.1.1.1 Kernel Machine Interface Lines -- Figure 45 illustrates the KM and how FMs attach to its buses. The signal lines are listed in Table 35, and briefly described as follows.

- "Source Address" lines select from among 256 locations. Some locations are assigned within the FU (see Table 36), but most are available for assignment to the attached FMs.
- "Ready" lines signify that the corresponding address or data lines have stable signals.
- "Source Data" lines carry the contents of the selected source registers.
- "Indirect" lines specify whether the selected operand registers contain pointers.

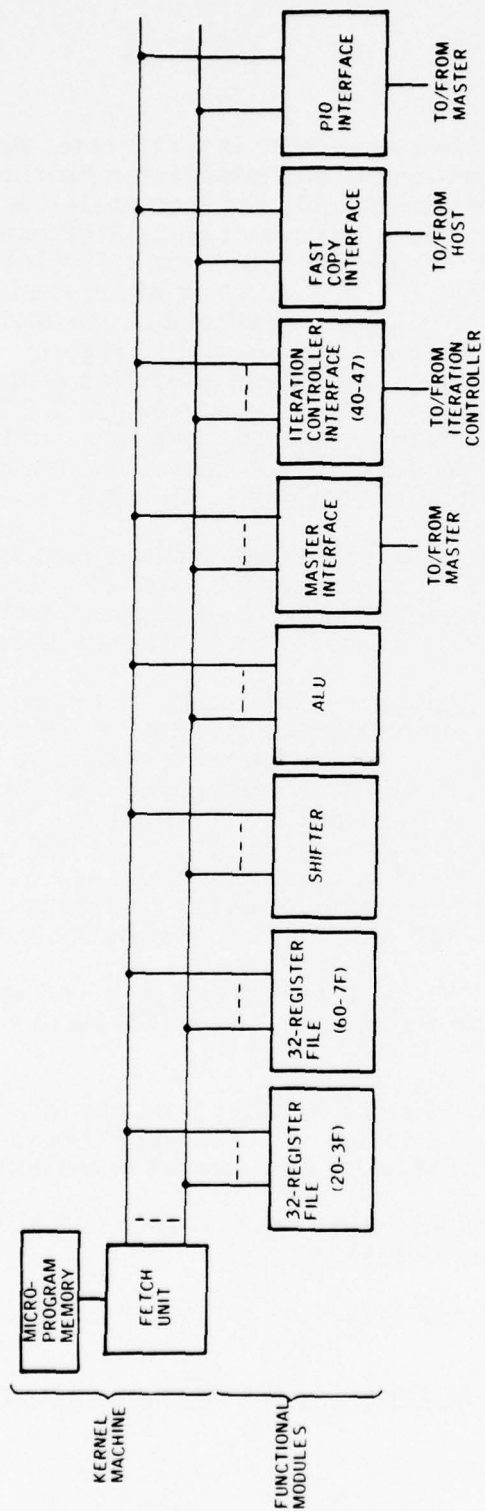


Figure 45. Interpreter Structure

Table 35. Kernel Machine Interface Lines

Signal Line	No. of Bits
Source A Address	8
Source A Address Ready	1
Source A Indirect	1
Source A Data	16
Source A Data Ready	1
Source B Address	8
Source B Address Ready	1
Source B Indirect	1
Source B Data	16
Source B Data Ready	1
Operator Select	6
Operation Select	6
OPOP Ready	1
Operator Ready	1
Operator Complete	1
Destination Address	8
Destination Address Ready	1
Destination Indirect	1
Destination Indirect Inhibit	1
Destination Data	16
Destination Data Ready	1
Destination Ready	1
Condition Inputs	31
Condition Input Enables	31
Abort	1
Error	1
Interrupt Request	1
Hard Interrupt	1

Table 36. Assigned Operand Addresses

SAA	SBA	DA	Assignment
00	00	---	Zero
---	---	00	No destination
01	---	01	Indirect Tag Memory
02	---	02	FU Write Protect Memory
03	---	03	FU Operator/Indirection Counters: OPC, SAIC, SBIC, DIC (left to right)
04	---	04	FU Condition Register bits 00-15
05	---	05	FU Condition Register bits 16-31
06	---	06	FU Condition Register bits 32-47
07	---	07	FU Condition Register bits 48-63
08	---	08	Microprogram Memory Data Register bits 00-11, right-justified, zero-filled
09	---	09	Microprogram Memory Data Register bits 12-27
0A	---	0A	Microprogram Memory Data Register bits 28-39, right-justified, zero-filled
0B	---	0B	Microprogram Memory Address Register
0C	---	0C	Microprogram Memory Address Stack Pointer
0D	---	0D	Microprogram Memory Address Stack. (top entry)
0E	---	0E	Unused
0F	---	0F	Unused

- "Destination Indirect Inhibit" is used by the selected operator to write directly into the specified destination register even though it contains an indirect pointer.
- "Operator Selection" lines select the FM operator to be used to perform an operation.
- "Operation Selection" lines select one of the operations that may be performed by the selected operator module.
- "OPOP Ready" signifies that the Operator/Operation lines carry stable signals.
- "Operator Ready" signifies to the FU that the selected operator is beginning to perform the operation.
- "Operator Complete" signifies that the operator has completed its operation(s).
- "Destination Ready" signifies that the selected location is prepared to receive information.
- "Abort" commands all active operators to immediately cease operation.
- "Error" notifies the FU that a module has detected a "fatal" error.
- "Interrupt Request" notifies the FU that some FM operator wants control of the machine.
- "Hard Interrupt" defines whether the interrupt being requested is to be handled by FM hardware or FU firmware.
- "Condition Inputs" and the associated "Enable" lines allow conditions generated by certain FMs to be tested by the microcode. The Enable lines cause the values present on the corresponding Condition Input lines to be stored in the FU Condition Register.

4.1.2.1.1.2 Fetch Unit Registers -- The principal FU registers have been divided into three groups for illustrative convenience:

- Register Bus Drivers (Figure 46)
- Memories and Associated Registers (Figure 47)
- Other (Figure 48)

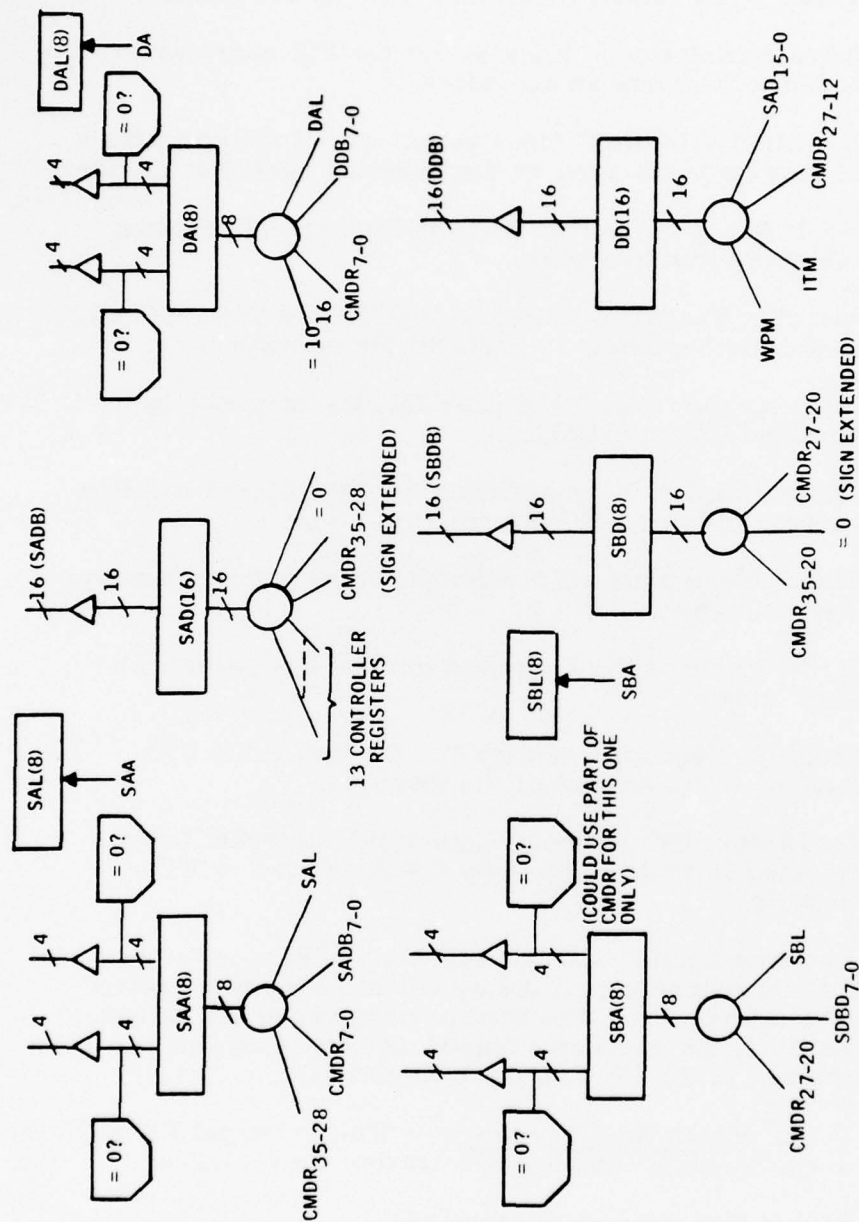


Figure 46. Fetch Unit Registers Primarily Related to Driving Register Busses

Register Bus Drivers: There are three register address/data bus pairs; the FU contains registers to drive each of these (FMs may also drive these lines). Since all FU registers are addressable (see Table 36), circuits are included to detect their addresses (which all have zero in the four most significant bits). Each FU register is connected to the input selector for the Source A Data register (SAD). Zero is provided as the data when the entire address is zero. Figure 46 shows these registers and their inputs.

Fetch Unit Memories: Four distinct memories are used in the FU:

- Microprogram Memory
- Microprogram Memory Address Stack
- Indirect Tag Memory
- Write Protect Memory

The Indirect Tag Memory requires three read access ports (see Section 4.1.2.1.1.3), so it is replicated three times; all three are written into simultaneously.

Figure 47 shows these memories and some associated registers.

Other Control Unit Registers: The majority of the other registers (Figure 48) hold conditions or counts which may affect sequencing or detect errors. The sequencer itself will not be specified at this time; it should not contain any registers except those used to record sequencing progress.

4.1.2.1.1.3 Microinstruction Classes -- Two microinstruction classes are provided:

- Operate
- Move, Test, and Branch.

Detailed descriptions of the instructions follow.

Microinstruction Format Type 0 -- Operate. - Type 0 microinstructions specify one or two source operands, an operation to be performed on the operand(s), and a destination for the result. The format is shown in Figure 49. The various fields will be described by following the flow of microinstruction execution.

Three bits define the Operand Addressing Mode (OAM); see Table 37. Until specified otherwise, the following description covers the case when the OAM is 4:*

*All codes, addresses, etc., are hexadecimal.

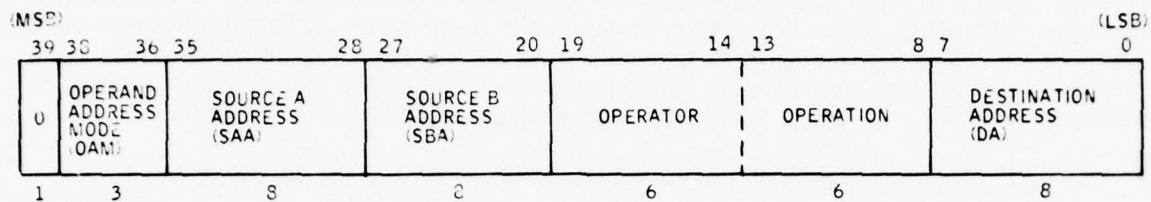


Figure 49. Microinstruction Format O -- Operate

Table 37. Operand Addressing Modes

OAM	Meaning
0	SAA, SBA are addresses; indirection determined by ITM.
1	SAA is an address, SBA is an 8-bit literal; indirection determined by ITM.
2	SAA and SBA are a 16-bit literal; indirection determined by ITM.
3	Read last link of indirect chain pointed to by SAA and/or SBA.
4	SAA, SBA are addresses; indirection-inhibited.
5	SAA is an address, SBA is an 8-bit literal; indirection-inhibited.
6	SAA and SBA are a 16-bit literal; indirection-inhibited.
7	Write last link of indirect chain pointed to by DA.

1) Source Operand Addressing. The KM has an operand address space of 256 locations or registers. Not all implemented registers need be accessible on all three operand buses, but the same address is reserved on all three buses for each register. However, every register must be accessible on at least one source bus and the destination bus to facilitate initialization and debugging. Certain operand addresses are assigned within the FU, as listed in Table 36.

Operand addresses are generated in three ways: they occur in microinstructions fetched from Microprogram Memory by the FU; they can be located by following a chain of registers containing "pointers"; and they may be emitted by operators during the course of an operation. A Programmed I/O (PIO) interface to another processor is such an operator; PIO is discussed in more detail in Section 4.1.2.1.1.5.

When an operand address is stable, the unit which generated it activates an appropriate Address Ready Line -- Source A Address Ready, Source B Address Ready, or Destination Address Ready. Each register in an attached FM must decode its own address. FU registers are selected by logic within the FU.

When a register detects its address on a source address bus, the selected register places its contents on the corresponding source data bus and activates the corresponding Source Data Ready Control line -- Source A Data Ready or Source B Data Ready. A zero source address is detected by the FU, which supplies a zero operand.

If the OAM is 4, this completes operand accessing. Alternate addressing modes include indirect chains and literals, and are described below.

2) Operator/Operation Selection. The 12-bit OPerator/OPeration (OPOP) code consists of a 6-bit Operator Select field and a 6-bit Operation Select field. The OPOP code can be generated in two ways: it may occur in a microinstruction, or it may be emitted by an operator during the course of an operation. In either case, an OPOP code can be viewed as performing a hardware procedure call within the currently executing microinstruction. The called procedure may be as elementary as a single Arithmetic Logic Unit (ALU) operation, or it may be a complex algorithm which emits sequences of source and destination addresses and additional OPOP codes. When an OPOP code signal is stable, the unit which generated it activates the OPOP Ready control line; the selected operator acknowledges this signal with a signal on the Operator Ready Control line. When an operator completes execution, it activates the Operator Complete control line.

Hardware procedures, like their software and firmware counterparts, may pass control among themselves in a hierarchical fashion. Each

operator which is capable of emitting OPOP codes has an Operator Counter which is initialized at zero. When the operator is invoked, it increments its counter by one. While the counter is nonzero, every subsequent OPOP Ready (by any operator) increments the counter, and every subsequent Operator Complete decrements the counter. The operator whose counter equals "1" is the lowest level one which is active: it has control of the machine and is the only operator which can drive the KM interface lines; its Operator Complete decrements its counter to zero, which disables it until it is invoked again, and returns control to the next higher level operator. The counters are four bits; attempts to overflow or underflow the FU counter will cause an error condition and set Condition Register bit 04 or 05 (see Section 4.1.2.1.1.4). The FU Operator Counter is an addressable operand (for diagnostic purposes), as may be those of the operators.

A machine may contain one Interrupt Operator module, through which operators can request their own invocation. When the Interrupt Operator receives an input signal from some other operator, it activates the Interrupt Request and Hard Interrupt control lines. At the completion of the current microinstruction, the FU executes the Activate Interrupt Module OPOP code (see Table 38) with zero source and destination addresses, which transfers control to the Interrupt Operator. The Interrupt Operator arbitrates its inputs and calls the requesting operator with the highest priority. There are two preassigned inputs to the Interrupt Operator: the PIO Operator, and the Control Panel Operator (see Sections 4.1.2.1.1.5 and 4.1.2.1.1.6). The Interrupt Operator may have additional inputs available as desired; the priorities of the PIO and Control Panel Operators are assigned by the designer in the context of the other operators which may cause interrupts. The Interrupt Operator also has facilities for enabling and disabling the input levels individually or as a group, and clearing the interrupt requests individually or as a group -- Table 38 lists the OPOP codes for each of these operations.

FMs can also invoke firmware routines by activating the Interrupt Request line with the Hard Interrupt line inactive. At the end of the microinstruction being executed when the Interrupt Request line becomes active, the FU will do a subroutine CALL to a microprogram memory address selected by special switches within the FU. Once at that location, the microcode must determine the cause of the interrupt by testing an FM register or the Condition Register.

If the microprogram memory address stack is full, the interrupt request will be disregarded until the completion of the next RETURN instruction.

Table 38. Preassigned OPOP Codes

Operator	Operation	Assignment
0	0	No operation.
0	1	Activate interrupt module.
0	2	Clear all interrupt requests.
0	3	Clear interrupt request whose number is the A operand.
0	4	Set interrupt mask to value of A operand.
0	5	Capture interrupt priority level.
0	6	Set interrupt priority level.
0	7	Read interrupt priority level.
0	8	Write microprogram memory.

All interrupt processing must either be done with all interrupts inhibited or else the interrupt handler (whether FM hardware or FU microcode) must stack the current priority level before establishing the new one. After processing the interrupt, the previous priority level must be restored.

Parameter passing among operators is by mutually-agreed-upon design conventions using the register address and data buses; the details depend upon the specific operator designs.

Each operator stores its source operand(s) internally, permitting the source buses to be used for accessing the other operands while the operator executes. When an operator has received its operand(s), it begins the operation. An operator which will issue further OPOP codes must copy the DA from the bus before it begins other actions if it wishes to place results in the specified destination register.

3) Destination Addressing. When an operation is complete, the operator places the result on the Destination Data Bus and activates the Destination Data Ready control line. If the operator has issued OPOP codes, it must retransmit the destination address before sending the data. If the operator wishes to use the register directly, independent of the bit in the FU's ITM, the operator must activate the Destination Indirect Inhibit (DII) line, which will prevent the FU

from sending any Destination Indirect signal. The operand register which decodes its address on the Destination Address bus sets a selection flip-flop when the Destination Address Ready control signal is present and the Destination Indirect control signal is absent. All other destination selection flip-flops are cleared by the Destination Address Ready control signal. If a register decodes its address on the Destination Address Bus and both the Destination Address Ready and Destination Indirect control signals are present, it transmits its contents on the Destination Data bus, sending the Destination Data Ready control signal when the data signals are stable. If the Destination Indirect control signal is absent, the destination register whose selection flip-flop is set will activate the Destination Ready control line when it is ready to accept data. The conjunction of Destination Ready and Destination data Ready causes the selected register to store the information from the Destination Data bus. The conjunction also causes the operator to deactivate Destination Data Ready. The Destination Ready line controls the rate at which the destination register receives data; it is not intended to accommodate varying destination register setup times, which are required to be small (this may require a high-speed buffer register in some modules).

Each unit capable of generating operand addresses contains a Write Protection Memory (WPM), defining which registers it is not allowed to write into. Unauthorized write attempts cause an error and set Condition Register bit 03 (see Section 4.1.2.1.1.4).

The WPM in the FU is itself an operand location which is always protected against FU writes, regardless of the corresponding protection indicator in the WPM. The same is true for the WPMs in all address-emitting operators except for the PIO Operator. The WPM in the PIO Operator may be simply a gate which permits PIO to write into any register. The method for writing into WPMs (where allowed) are described later in this section under "Write Protection."

The completion of an operation or microinstruction is detected by the conjunction of Destination Ready, Destination Data Ready, and the Operation Counter of the unit being one. The next action is then performed; if the FU is in control, the completion signifies the completion of a Type 0 instruction, which causes a test for interrupt requests; if none are present, the microprogram memory data register is loaded with the next microinstruction.

The general discussion of the execution of Type 0 instructions is now complete. Numerous variants and special cases can modify this sequencing, as we now describe.

4) Reserved Values. A zero OPOP code designates no operation. A zero Destination Address corresponds to no register; it is used

when the result of an operation is not to be stored (for example, when the result is only to be tested, as occurs with Type 1 microinstructions). The FU controls the Destination Ready line when the Destination Address is zero or when the destination is a FU register.

5) Constants. There are two ways to introduce constants. The first is to store the constant in a register which is a write-protected (as required). The advantage of this is that constants are treated like all other operands. The disadvantage is that the constants occupy register space which may be needed for variable operands.

The second means of introducing constants is to place them in microinstructions as literal quantities. Short constants are specified under Operand Addressing Mode 1 or 5, in which the information on the Source B Address bus is an 8-bit literal consisting of seven value bits and a sign bit. The FU transfers the literal to the Source B Data bus in right-justified and sign-extended form and activates Source B Data Ready. This approach accommodates the most commonly used numeric constants without consuming registers.

Long constants are specified when the OAM is 2 or 6; a full 16-bit literal is located in the pair of source address fields, with the low-order byte on the B side; the FU transfers the literal to the Source B Data Ready. If the Destination Address is nonzero, the FU copies it onto the Source A Address bus to obtain the A operand; this register is also used as the destination in the usual manner. (Note that this requires the destination register to also be attached to the Source A Bus.) If the Destination Address is zero, the A operand is zero, and the result of the operation is not stored. This approach to literals avoids registers, and provides for logical and large magnitude numeric constants; however, it is less flexible than the byte literal mechanism.

6) Indirection. Thus far, we have assumed that all operand register references were direct (i. e., addresses of registers which contain the operand). However, the FU also provides for multilevel indirect operand addresses (i. e., addresses of registers which themselves contain addresses rather than operands). Indirection is specified by a tag associated with each operand register; this frees the microcode and operators from the necessity of knowing whether references are indirect. Operand Address Zero (containing the zero operand) is always direct.

Indirection may occur when the OAM is 0, 1, or 2. The procedure is as follows. If a referenced operand's indirect tag is set when its Ready Line is active, the referencing unit (FU or operator) changes the operand address to that in the low-order byte of the corresponding data bus. Indirections are resolved on all three operand buses simultaneously; each non-overlapped level adds one FU clock period to the microinstruction execution time. Up to 15 levels of indirection

are allowed in each of the operand addresses in a single microinstruction; an attempt to exceed this limit causes an error condition and sets Condition Register bits 06 or 07 (see Section 4.1.2.1.1.4).

The three operand addresses will be direct, regardless of the values of the corresponding indirect tags, if the OAM is 4, 5, or 6 (normal addressing, byte literal, and word literal, respectively). This allows the microcode to read or change pointers without having to first clear and then restore the corresponding indirect tags.

It is also possible for the microcode to directly read or write the last pointer in an indirect chain by setting the OAM bits to 3 or 7, respectively. Literals cannot be used, nor can indirection be inhibited. When OAM is 3 and both sources are direct, or when OAM is 7 and the destination is direct, an error condition occurs and Condition Register bit 08 is set (see Section 4.1.2.1.1.4). Special operators for more-complex list manipulation may be implemented as operator modules connected to the buses if required.

Instead of being physically located with their corresponding registers, the indirect tags are gathered in an FU Indirect Tag Memory (ITM). The ITM has 256 one-bit words, each of which specifies whether the corresponding register currently contains a pointer or an operand. Any appearance of an operand address on the address buses causes an ITM access; the ITM has three read ports to allow all three operand addresses to be checked simultaneously. The ITM outputs are control lines called Source A Indirect, Source B Indirect, and Destination Indirect. The Destination Indirect signal will not be activated if the FU is receiving the Destination Indirect Inhibit signal from an operator, or if the OAM bits inhibit indirection and the FU's Operator Counter contains "1".

The FU also uses the Indirect control lines to reflect in bits 0C and 0D of the Condition Register whether the operand addresses specified in the current microinstruction are direct or indirect (literals are direct), independent of whether the OAM bits inhibit indirection. Subsequent operand addresses supplied by the specified operator do not affect these Condition Register bits. This provides a convenient means of testing whether a register contains an operand or a pointer, especially by use of the Type 1 microinstruction.

The value of a specified ITM bit may be read and stored as well as tested. To read the ITM, it is addressed as a source on the Source A Address bus, with the desired bit number on the Source B Data bus. The bit value is placed in the least-significant bit (LSB) position of the Source A Data bus. The bit number may be known to the microcode and expressed as a byte literal or it may be unknown (e.g., indexed) and be supplied from a register specified by the Source B Address.

Because the ITM cannot be correctly accessed until the Source B operand is available, the ITM does not activate Source A Data Ready until Source B Data Ready is active and Source B Indirect is inactive; thus, reading the ITM is one FU clock period slower than testing it. The ITM address is forced by the FU to be direct, regardless of the value of the corresponding ITM bit.

To write into an ITM bit, the ITM is addressed as a destination, with the desired bit number in the low-order eight bits of the Destination Data bus, and the value to be written in the most-significant bit (MSB) of the Destination Data bus. Some operator (such as a "move" operator) is required to appropriately position the contents of the source data buses on the Destination Data bus. As when reading, the ITM cannot be written until a bit number is available; the ITM does not activate Destination Ready until after Destination Data Ready is active; thus, writing into the ITM is also one clock period slower than testing it.

7) Write Protection. If a unit is sending data to a destination, the WPM in that unit is consulted to determine whether the write is legal. This test is made by using the Destination Address signal to select the WPM bit. However, if the module is using a Destination Address as a component of an indirect addressing chain, the WPM bit is ignored, since that register is not actually receiving data.

A WPM is read in about the same way as the ITM: the output of the FU WPM is also input to Condition Register bit 0E when read. Other WPMs may direct their outputs to the KM's Condition Register; this is a module design issue. Writing a WPM is also nearly the same as writing the ITM. As usual, the write protect mechanism of the module controlling the bus governs whether or not the write to a WPM is legal.

8) Hangups. A timeout mechanism is used to detect illegal addresses and operator/operation codes. Whenever an Address Ready signal or Operator Ready signal is detected by the FU, a timer is activated. If the timer times out before the corresponding Register Ready or Operator Ready signal is detected, an error condition is signalled and Condition Register bit 01 or 02 is set (see Section 4.1.2.1.1.4).

Microinstruction Format Type 1 -- Move, Test, and Branch. - Type 1 microinstructions move information, test Condition Register bits (which include the values of the moved information), and perform microinstruction sequence modifications. Four types of sequence changes are possible: GOTO, CALL, IF-THEN-ELSE (ITE), and RETURN.

Figure 50 depicts the format of all Type 1 instructions. Generally, the fields are used as follows:

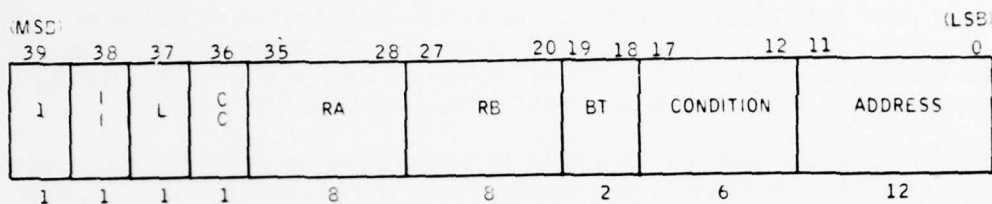


Figure 50. Type 1 Instruction Format

- II specifies inhibition of indirection on the RA and RB fields.
- L specifies that the RA field contains a byte literal.
- CC specifies that the condition is to be tested for 0 rather than 1.
- RA and RB specify registers.
- BT specifies the branch type: GOTO, CALL, ITE, or RETURN.
- COND selects the Condition Register bit to be tested.
- ADDRESS specifies the branch address (in case of GOTO, CALL, and ITE); it is not interpreted in RETURN instructions while the FU is operating in Normal mode (see Section 4.1.2.1.1.4).

The general sequence of operation for Type 1 microinstructions is the following:

- 1) Find the effective RA (if L = 0) and RB addresses. The RB address is used as a destination during this instruction.
- 2) Copy the contents of the A target register or the literal value (if L = 1) to the destination register. If the RB field is zero, the A value will not be copied anywhere, but will be used to change Condition Register bits (this will affect some condition values). However, if both RA and RB are zero, no value is copied and no Condition Register bits are changed. The FU uses the Source A and Destination buses to make the move; internally the FU copies the Source A Data to the Destination Data bus.
- 3) Test the condition and select the next microinstruction (which, in the case of CALL and ITE, requires pushing the return address onto the Microprogram Memory Return Address Stack).

Branch conditions are selected from among condition lines available to the FU in a predetermined way. Some of the conditions are generated and selected by circuitry within the FU; others are generated by FMs attached to the buses and to the condition lines. Table 39 lists the conditions assigned and detected within the FU.

The 31 unassigned Condition Register bits may be set by FMs attached to the KM. Two lines are associated with each of these condition bits -- the Condition Value line and the Condition Enable line. An activated Condition Enable line causes the value on the corresponding Condition Value line to be stored in the Condition Register bit. This bit may be tested by Type 1 microinstructions; the FU hardware does not otherwise modify its execution sequences based on the values in these 31 condition bits. (Possible uses for these condition bits include arithmetic overflow indication, and FM or external control signals; the exact assignments depend upon the application.)

The specifics of selecting the next microinstruction depend upon the branch type; they are discussed below.

1) GOTO. If the condition is true, the GOTO is performed by copying ADDRESS from the microinstruction to the Microprogram Memory Address Register. If the condition is false, the next sequential microinstruction is executed.

2) CALL. If the condition is true, the address of the next sequential microinstruction is pushed onto the Microprogram Memory Return Address Stack, and a branch to the address specified in the CALL instruction is made. If the condition is false, the next sequential microinstruction is executed. (Note that this constitutes the structured programming IF-THEN construct.) The Microprogram Memory Return Address Stack allows subroutine nesting to 16 levels; when nesting reaches 15 levels, the Condition Register stack almost full bit (0A) is set. If the application microcode is such that the stack may become filled, this bit should be tested prior to the CALL; the remaining location in the stack permits this test also be performed with a CALL microinstruction. Attempting to perform a CALL when all 16 stack locations are filled will cause an error condition with Condition Register bit 09 set (see Section 4.1.2.1.1.4).

3) IF-THEN-ELSE. One of two subroutines is called, depending on the Condition Register bit: if the condition is true, the entry address is taken from the ITE instruction; if the condition is false, the entry address is taken from the next sequential microinstruction (the remainder of that word is ignored). Either subroutine returns to the second microinstruction following the ITE instruction. Stack overflow is handled the same as for CALL.

Table 39. Condition Register Bits Assigned by the Fetch Unit
(Bits 00-09 are reset when tested)

Condition Bit (hex)	Assignment (Bit=1 if Condition True)
00	External error signal
01	Register timeout elapsed
02	Operator timeout elapsed
03	Unauthorized write attempted
04	Operation counter underflow
05	Operation counter overflow
06	Source indirection counter overflow
07	Destination indirection counter overflow
08	LAST mode used with no pointers in register
09	CALL attempted with Microprogram Memory Return Address Stack full (=16)
0A	Microprogram Memory Return Address Stack almost full (=15)
0B	Microprogram Memory Return Address Stack empty
0C	A source was indirect
0D	Destination was indirect
0E	Last destination WPM bit from FU
0F	Destination Data = 0
10-1F	Destination Data Bus Bits 0-15
20	Always = 0

4) RETURN. If the condition is true, the Microprogram Memory Return Address Stack is popped and the address that is removed is placed in the Microprogram Memory Address Register. The ADDRESS field of the microinstruction is not used (but see Section 4.1.2.1.1.4). If the microcode is such that there is a possibility of stack underflow, the Condition Register stack empty bit (0B) should be tested prior to the RETURN. Attempting to RETURN with an empty stack causes an error condition (see Section 4.1.2.1.1.4).

4.1.2.1.1.4 Errors -- Error conditions may be caused by a machine condition detected by the FU or by a signal emanating from an FM connected to the KM. Error signals from FMs are ORed to produce a single FM error signal. (If the system designer wishes to discriminate among the FM-caused errors, he must use external inputs to the Condition Register or define them in some other FM register.)

The FU operates in one of two modes: Error mode and Normal mode. Errors and interrupts are not detected while the machine is in Error mode, although they can be found by testing the Condition Register. The FU enters Error mode whenever an error condition is detected; it reenters Normal mode by executing a RETURN instruction bit 0 = 0. This allows the machine to return to Normal mode as soon as possible after the error has been detected, even though the error handler may not have completed execution and returned control to the interrupted routine. Details are provided in the following paragraphs.

Every machine-detected error condition is associated with some bit in Condition Register positions 00-09 (these conditions are listed in Table 39). Whenever any internal error bit is first set (note that if an error bit remains set, no error actions occur -- to permit error handling on a priority basis), all operations currently in progress are terminated by the FU, which emits a signal on the Abort control line. The Abort signal resets the Operator Counters in all operators attached to the KM and sets the FU's Operator Counter to 1, thus prohibiting all operators from emitting bus control signals and placing the FU in control.

The FU now enters Error mode and behaves as though a subroutine whose entry point is located at a switch-selectable microprogram memory location is called: the current Microprogram Memory Address Register value is saved in the Microprogram Memory Address Stack, and the Microprogram Memory Address Register is then loaded from special switches within the FU. If the Microprogram Memory Return Address Stack was full before the error was detected, the oldest entry will be lost and the more recent entries will be saved. The stack full condition bit is not cleared by this action; then the error-handling microcode can test whether stack overflow occurred. Note that if stack overflow occurred there is no way to return from the subroutine nest. This occurrence is fatal; the recovery routine will have to abort everything.

The error handling routine at the switch-selectable location will test the Condition Register or some F'M register to determine the specific error condition and branch to the appropriate handler.

Normal mode operation is resumed when the FU executes a RETURN operation whose least significant address bit is zero. However, the existence of any set bit in Condition Register positions 00-09 cause an immediate reentry to Error mode without affecting the Microprogram Memory Return Address Stack.

4.1.2.1.1.5 Programmed Input/Output -- As used here, the term "Programmed Input/Output" (PIO) refers to the capability for the Master mini-computer to obtain control of the Slave Controller to initialize it, read its status, diagnose it, etc. All Slave access is done through the Interpreter. PIO is a straightforward instance of an operator which can request its own invocation through the Interrupt Module.

4.1.2.1.1.6 Control Panel -- The requirements for a control panel are as yet undetermined. With few conceivable exceptions, the control panel functions can be handled by making the control panel an operator.

4.1.2.1.2 Functional Modules for the ECAM Interpreter -- This section describes the functional characteristics of the modules that will be connected to the KM to construct the Interpreter and to connect to the other subunits of the Slave. The modules are divided into three categories:

- Register modules
- Operator modules
- Interface modules

Each module has been arbitrarily assigned addresses and operation codes; these assignments could be changed in any way compatible with the reserved assignments described herein.

4.1.2.1.2.1 Register Modules -- Two modules containing register files are used in the Interpreter. One interface module (Section 4.1.2.1.2.3) also has register addresses.

Each register file module contains 32 registers, each 16 bits wide. Each register file interfaces with one source bus and the destination bus. The only actions taken by the register files are reading and writing individual registers; these activities do not imply further action.

The two modules are specified in Table 40.

Table 40. Register Module Specifications

Item	Register File 1	Register File 2
Register Addresses (hex)	20-3F	60-7F
Source Bus	A	B
Destination Bus	Yes	Yes
Operations	Read/ Write	Read/ Write
Function	32 Registers/ 16 bits ea.	32 Registers/ 16 bits ea.

4.1.2.1.2.2 Operator Modules -- The Interpreter does not require extensive arithmetic capability. It does require logical operations and shifting for instruction field isolation. Two modules are described, one for shifting and the other for arithmetic and logic, only because in currently available technology the ALU and shifting functions are offered in different chip sets. When the machine is built, however, it may be desirable to combine these functions into one module, thereby effecting some savings in the bus interface logic.

Both Operator modules interface with both sets of Source Data lines, the Destination Data lines, and the OPOP lines. Register Address lines and the associated control lines are not used by these modules. In the following descriptions, A, B, and D are used to denote the values on the A, B, and D data lines, respectively.

The ALU has 10 functions, listed in Table 41. All arithmetic functions produce a carry/borrow result that is held within the ALU module; this bit is called C in the descriptions. The ALU module transmits C to the controller's Condition Register at the completion of each ALU operation. Bit 23 of the Condition Register is assigned to receive this information.

The 12 leftmost bits of B are ignored during short multiply; no overflow or error condition results if these bits are nonzero.

No error or interrupt signals are caused by the ALU. It does not issue OPOP codes itself.

The ALU is assigned OPOP codes $1/0_{16}$ - $1/9_{16}$. Operator code "1" selects this module.

Table 41. ALU Functions

OP/OP Code	Function	Action
1/0	Add	$D = A + B$
1/1	Add with Carry	$D = A + B + 1$
1/2	Add with Saved Carry	$D = A + B + C$
1/4	Subtract	$D = A - B$
1/5	Subtract with Borrow	$D = A - B - 1$
1/6	Subtract with Saved Borrow	$D = A - B - C$
1/3	Short Multiply	$D = A * B_{3-0}$
1/8	AND	$D = A \wedge B$
1/9	OR	$D = A \vee B$
1/7	NOT	$D = \neg A$

Assume, for purposes of functional description (but not necessarily the implementation), that the shift unit has two internal 16-bit registers, S and T. These are connected as a simple bidirectional shift register, with T to the right of S. The operation of the shift unit is given by the following algorithm:

- 1) Copy A to S and B to T or A to T and B to S; the latter is performed when the Operator Code is 3.
- 2) Shift S, T by the amount specified (in the Operation Code) in the direction specified (in the Operation Code).
- 3) Copy T to D (the result).

Shifts are allowed in either direction with a maximum 31 shifts possible. All shifts fill with zeros. (Note that left shifts of more than 15 positions produce a zero result.)

The shift unit does not produce any errors, interrupts, or condition values. It does not issue OPOP codes itself.

The shift unit is assigned OPOP codes 2/0-3/3F. (The six operation bits are the direction and shift count, while the operator select of "2" or "3" selects this module, and the choice of operator number determines how the A and B data are loaded before shifting.)

4.1.2.1.2.3 Interface Modules -- Each interface module has different characteristics which are heavily dependent upon the modules at the other side of the interface. Four interfaces are required for the slave controller:

- Master Interface
- Iteration Controller Interface
- Fast Copy Interface
- PIO Interface

Note that though each of these modules is serving as an interface, it still appears from the slave side as a register module or an operator module. The interface characteristics manifest themselves in the actions taken when the module is selected by the controller.

The interface modules are discussed below.

1) Master Interface. The Master Interface is viewed as an operator module from the controller. Therefore, it is connected to all Source and Destination Data buses and to the OPOP lines. Its three operations are:

- Read Master Memory (code 4/2)
- Write Master Memory (code 4/0)
- Send Interrupt to Master (code 4/1)

For all three operations, the Source A value is interpreted as a master memory address. The Source B value is ignored for READ, and used as a data word for WRITE and INTERRUPT. READ produces a result on the Destination Data bus which is the contents of the word selected from the master's memory; the other operations do not produce data; they are terminated after the interface module has buffered the request. Request buffering may incur a delay if the interface module is still handling a previous request; this delay is controlled by the module not sending an Operator Ready Control signal until it is ready to buffer the following operation request.

The READ and WRITE operations perform obvious functions. INTERRUPT performs a write using the address and data from the A and B inputs, respectively, before transmitting the interrupt signal to the master. This Write can be used to transmit status information to the master's interrupt handler.

The interface module remains busy after initiating an interrupt action until the interrupt signal has been received by the master's processor.

This interface module does not send condition or error signals to the controller.

2) Iteration Controller Interface. The Iteration Controller, or IT, interface is viewed as a set of registers from the controller. It is connected to the Source A and Destination Data buses.

This module has eight register addresses, with the assignments shown in Table 42. Three of the registers are read-only; they present status and count output information from the array. Three other registers are read/write in conventional ways. The two remaining registers are read/write, but reading or writing into them causes other actions to be initiated, as described below.

Table 42. IT Interface Register Assignment

Register Address (hex)	Width	Name	Read	Write
40	12	Length	X	X
41	13	Parameter Buffer Address	X	X
42	16	Parameter Buffer Data	X*	X*
43	8	Instruction Code	X	X*
44	1	Buffer Select Bit	X	X
45	16	IT Output ₀₋₁₅	X	--
46	16	IT Output ₁₆₋₃₁	X	--
47	16	Array Status	X	--

*See text for explanation of implied operation.

The IT interface places IT commands and data in buffer memory locations; upon signal, the command is performed using the supplied data. There are two identical sets of buffer registers -- Set 0 and Set 1. Each set contains registers holding length, parameter buffer address, and instruction code. In addition, each set is associated with a random-access memory (RAM) holding 8192 sixteen-bit words. The Buffer Select Bit selects which set will be used.

The Parameter Buffer Address and Data registers are used to access the RAMs within the interface. These registers are not the actual address and data registers associated with the memory, however, since the IT must also have access to the memory. Accessing the Parameter Buffer Data Register causes a corresponding access to the memory location selected by the Parameter Buffer Address Register, followed by an incrementation of the Parameter Buffer Address Register. When data are written into the Data register, they are also written into the selected memory at the selected address. When the Data register is used as a

source, the value is obtained by reading from the memory (and then incrementing the address). The source data will not be available until the memory has placed the selected word in the Data register, from which it is placed on the Source A Data bus.

Writing into the Instruction Code register causes the interface module to set an occupancy bit associated with the register; the IT may perform an operation whenever the occupancy bit associated with either Instruction Code register is set. If both are set, the interface module will not signal the IT about the last one that was set (until the first has been cleared). The Code register occupancy bits are cleared by the IT upon completion of the operation held in the register. In addition, the interface sets Condition Register bits 21 or 22 when the instructions in buffers 0 or 1, respectively, are completed. The Condition Register bits are reset when the Instruction registers are loaded.

This interface module controls two condition bits in the KM; it does not emit error signals. It uses eight register addresses: 40-47₁₆.

3) Fast Copy Interface. The Fast Copy Interface controls the mechanism for quickly copying information between the host machine and the array. This mechanism has not been designed in detail during this contract, but can be specified at the interface level. It will behave like an operator in which control words are sent to it on the Source A and Source B Data Lines. The outputs to the KM control unit include status information regarding whether the fast copy unit is active. There are no error conditions detected in this module.

4) Programmed I/O Interface. The PIO Interface is used by the master machine to set and read registers within the slave. This action may cause the slave to initiate a new sequence of events, or may be used to test the health of the slave machine. General details of PIO modules are given in Section 4.1.2.1.1.5.

5) Module Summary. The register address assignments and OPOP Code assignments for the slave machine are summarized in Tables 43 and 44. Condition Register bit assignments are specified in Table 45. Assignments for the Fast Copy Interface are not included in the tables.

4.1.2.2 Iteration Controller -- The function of the IT is to transform the commands it receives from the interpreter into sequences of signals causing the word logic to perform the specified operation on all bits of a field.

Table 43. Register Address Assignments

Register Address (hex)	Register Location
00-0F	Kernel Machine
10-1F	Unused
20-3F	Register Module A
40-47	IT Interface
48-5F	Unused
60-7F	Register Module B
80-FF	Unused

Table 44. OPOP Codes for the Slave Controller

OPOP Code (hex)	Function	Mode Activated
0/0-0/8	Miscellaneous	Kernel Machine
0/9-0/3F	Unassigned	
1/0-1/9	Arithmetic	ALU
1/A-1/3F	Unassigned	
2/0-3/3F	Shift	Shift
4/0-4/2	Signal Master	Master Interface
4/3-3F/3F	Unassigned	

Table 45. Condition Register Bit Assignments for Slave Controller (See Table 39 for bits 00-20)

Condition Bit (hex)	Assignment
21	IT Instruction from Buffer 0 Completed
22	IT Instruction from Buffer 1 Completed
23	ALU Carry Bit
24-3F	Unassigned

The interface between the Interpreter and the IT is via a pair of dedicated buffer memories, as shown in Figure 51. Each memory contains an 8-bit IT-code register used to pass the instruction, a 12-bit register in which field lengths are specified, and an 8K (twice word size) parameter buffer. The status of the buffers is maintained by a "ready" flip-flop in each one. The ready flip-flop is set when an IT code is entered in the register by the Interpreter, and is reset by the IT when the buffer has been processed. The IT responds to the first buffer which becomes "ready," latching up a buffer select signal which causes the appropriate paths to be connected.

The IT is internally microprogrammed to emit two types of control sequences to the array, as shown in Figure 52. The first is straight-line execution of a series of micro-operations within the array. This structure is used for operations such as match-bit transfers which are not repeated over a number of bits.

The looping structure is primarily used for operations (such as searches) which are repeated a number of times. The structure allows a block of setup instructions, followed by a block which is repeated until a loop exit criterion is satisfied. Either of two exit criteria may be used: "end-of-field" based on a counter, and "no-activity" where the activity indicator is true when any match bit (m) in the array is true.

The microinstruction formats which are used to code sequences of IT operations are shown in Figure 53. Type 0 instructions are the ones which cause operations in the array and are the principal ones used. Type 1 instructions cause internal IT operations and are mainly provided for maintenance and diagnostic purposes.

Within the Type 0 instruction, the first three fields are used directly to cause array operations and the last two control IT sequencing. The "code" field contains the value to be transmitted as the array function code, the S field determines the storage operation to be performed, and the P field controls parameter transfers.

Looping in the IT microprograms is controlled by the L field of the Type 0 instruction. For nonlooping sequences, all microinstructions except the last have a "0" in bit 7 of the word. The last microinstruction contains the "111" unconditional termination code. Loops are created by marking the beginning of the loop with an L-code of "0x1" and using either "100" or "101" in the L field of the last instruction. Instructions preceding the first one of the loop have "0" in bit 7. Instructions within the loop have L-code "0x0."

The H (hold) field of the Type 0 instruction specifies the length of time the microinstruction is to remain in execution. This is done by specifying a hold count of 1 to 31 bit-times. In the baseline, this is 1 to 31 microseconds.

Type 1 instructions are used to map the 8-bit instruction code received from the Interpreter into a longer code to transfer data among the slave's registers. Details of this control code format will be determined during logic design.

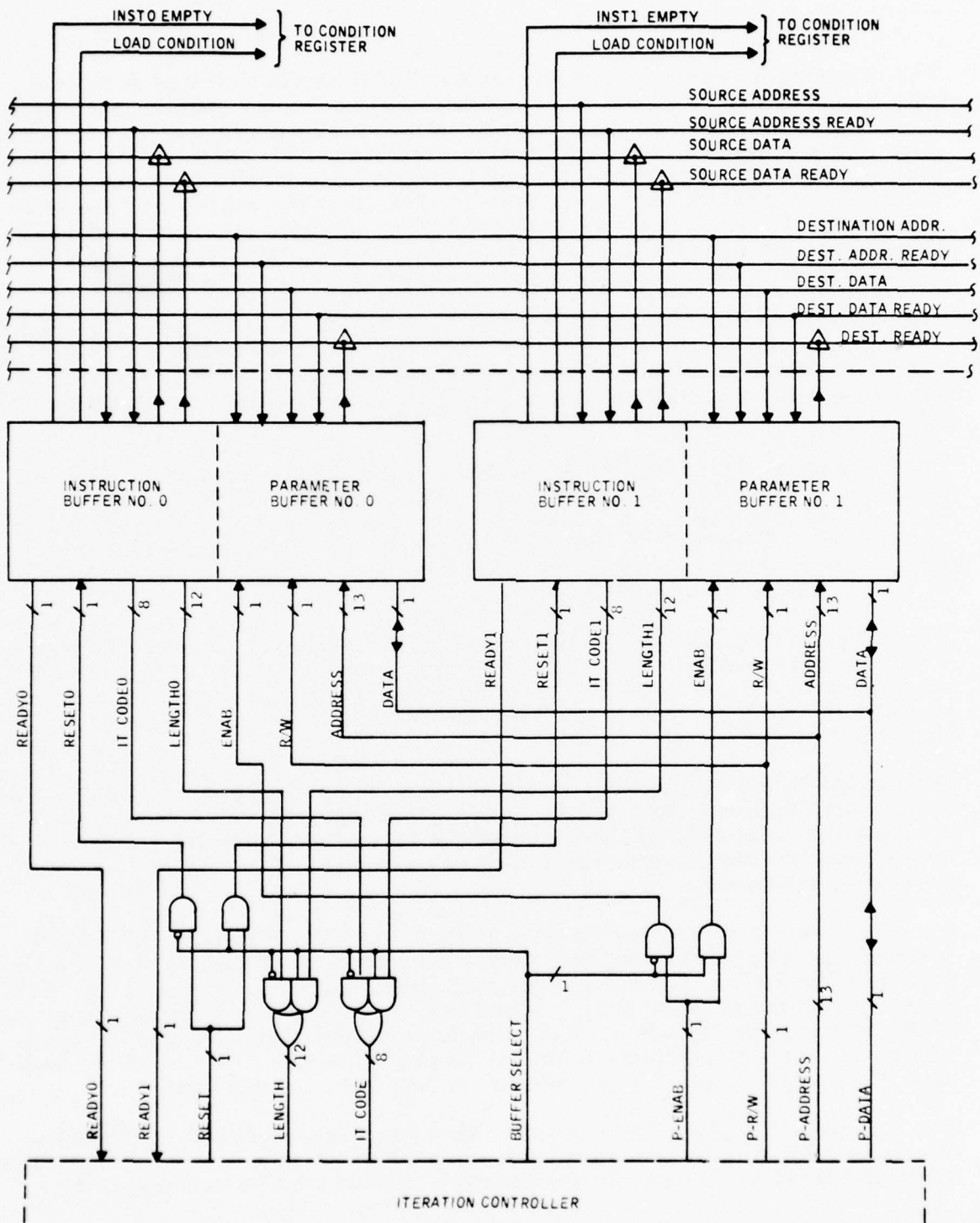
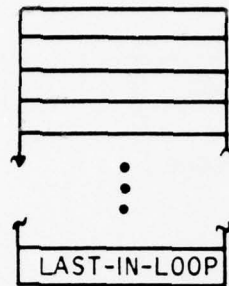


Figure 51. Parameter and Instruction Buffering

(A) NO LOOP



(B) LOOP

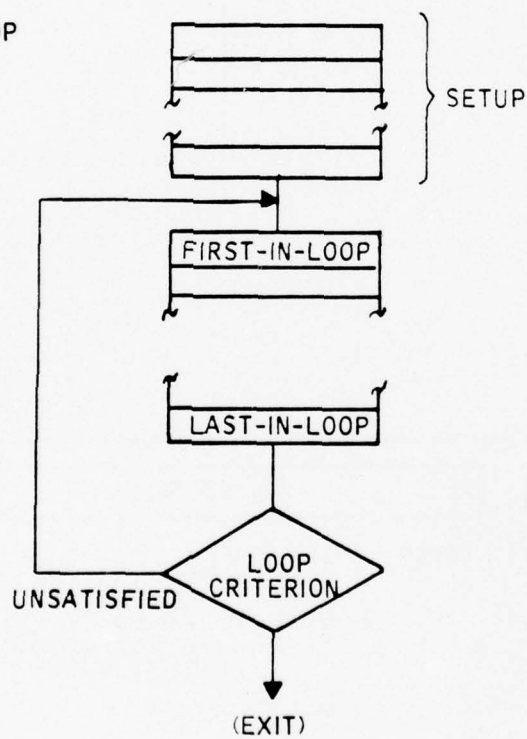
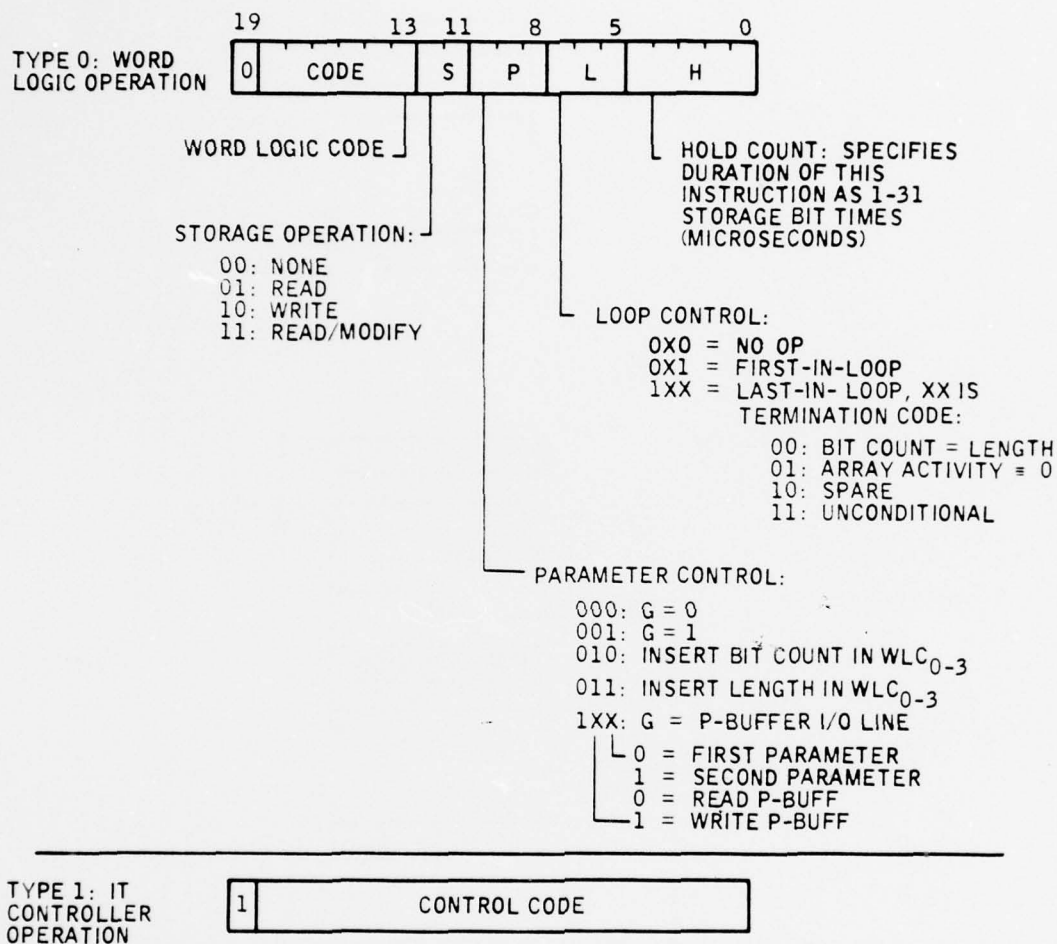


Figure 52. Iteration Control Sequences



- CODES:
- "TO": SHIFT STORAGE TO BIT POSITION SPECIFIED IN LENGTH REGISTER.
 - "READ": COPY IT CONTROLLER REGISTER (SPECIFIED) BY ADDRESS IN LENGTH REGISTER) INTO IT CONTROLLER OUTPUT REGISTER.
 - "MASTER CLEAR"

Figure 53. IT Controller Microcode Formats

The register-level structure of the IT is shown in Figure 54 and the control sequence is shown in Figure 55. The IT control sequence is separated into four major functions: Interpreter Buffer access, Type 0 execution preparation, execution control operations, and Type 1 execution. These are described below.

Interpreter Buffer access is made when an instruction-ready signal is received from either of the Instruction buffers. When this signal is sensed, the identity of the Ready Buffer is latched in a flip-flop, causing the steering logic of Figure 51 to make the proper connections. (The way in which the interpreter handles the buffer guarantees that no races can occur.) Once the buffer has been selected, the upper six bits of the IT code are used as the upper six bits of a 9-bit microprogram address, with zeros as low-order bits. This primitive hashing is adequate to scatter the short sequences of microoperations evenly through the storage. Where more than eight microinstructions are required for a given operation, the next succeeding code is left unused.

Assignment of codes to the IT functions of Figure 53 is a trivial exercise that has been left until microprogramming is done.

Execution preparations for Type 0 instructions use the lower two bits of the IT code, together with the S field of the current microinstruction and the mask portion of the Parameter Buffer. The first test made determines whether the current microinstruction includes a storage operation. If it does not, masking and storage control tests are skipped. If it does, and the IT code specifies that a mask is to be used for storage operations, then the bit is accessed and tested. The P-buffer address of this bit is formed by left-concatenating a "1" onto the bit-count register, thus accessing the upper 4K bits of the buffer. If the operation is masked in this bit position, a NO-OP is selected as input to the Word Logic Code (WLC) register. If not, the code from the microinstruction is selected.

Once the proper code has been determined, the IT next requests access to the appropriate bit (from BIT ADDRESS). When the Storage Controller makes the bit available, the completion status of the previous array operation is checked, and newly-prepared code is issued ("triggered") for execution. This is done by loading the WLC, Store Code, and Hold Count registers and enabling the storage controller. Once these things have been done, timing of the operation is independent of the IT and is based on the Hold Count. When the time required for the operation has elapsed, the WLC register is cleared to NO-OP status and the "Array Operation Complete" indicator returns to true.

While the issued instruction is executing, the IT performs a number of execution control operations to determine the next microinstruction to be fetched. If a storage operation is being performed, the BIT COUNT is incremented to reflect the activity and the (next) BIT ADDRESS is obtained by incrementing or decrementing the register. Next, the L field is checked to determine loop control actions. If the instruction is the first in a loop, its address is

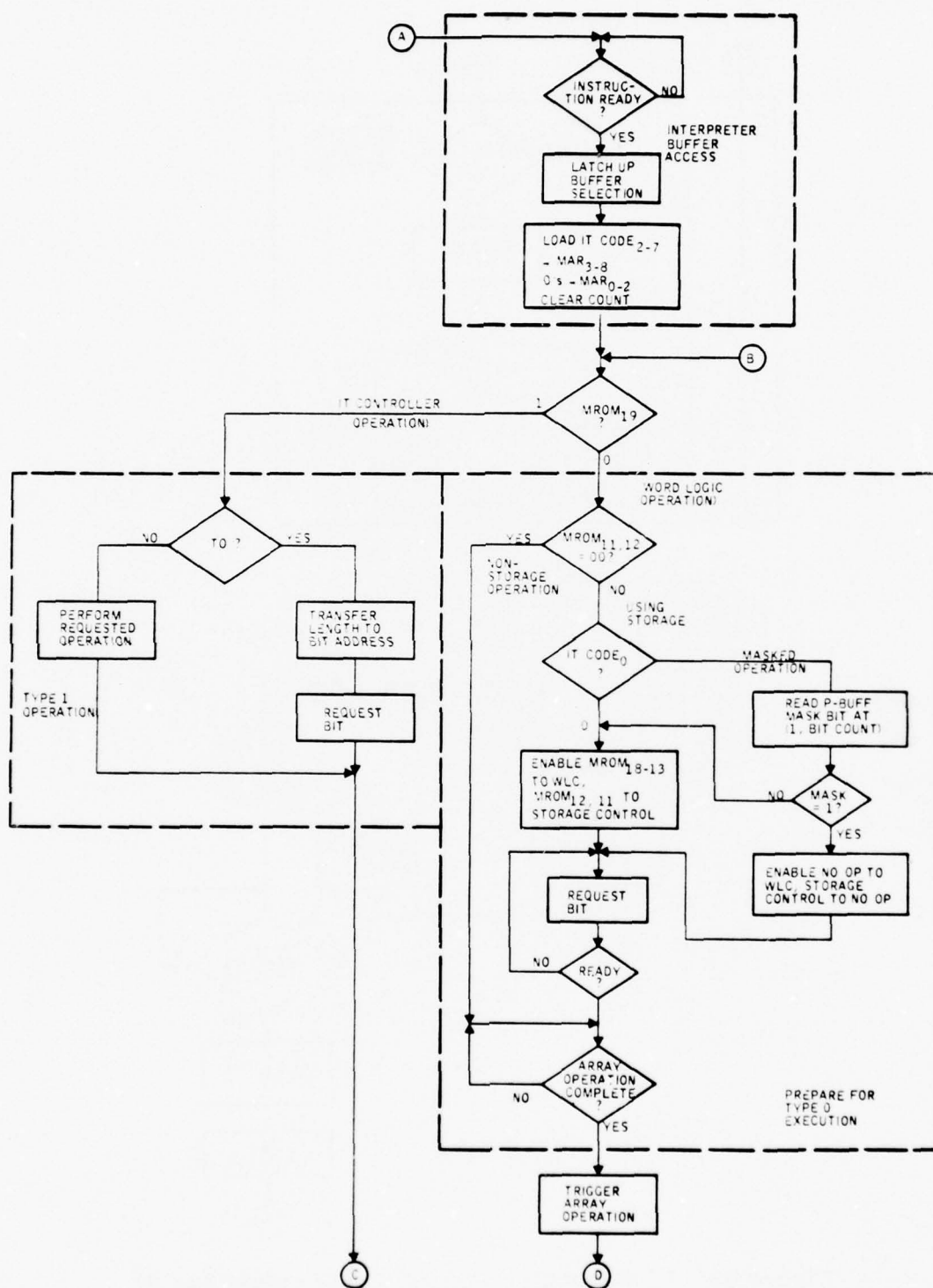


Figure 55. IT Controller Control Logic (Sheet 1 of 2)

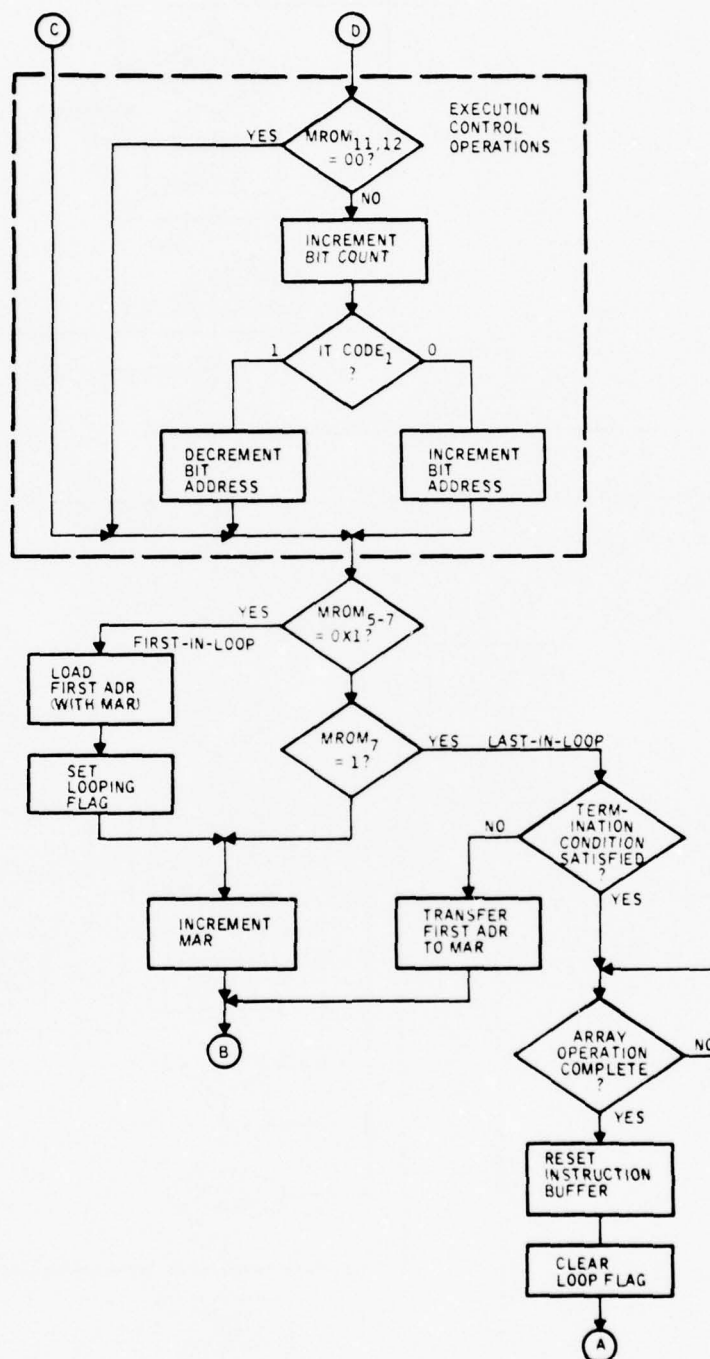


Figure 55. IT Controller Control Logic (Sheet 2 of 2)

saved and the Memory Address register is incremented. If it is a "middle," only the increment is performed. If it is a "last," then the loop exit criterion is checked to determine whether the execution should be terminated or the next instruction should be from the previously-saved "first" address.

Type 1 instruction execution distinguishes between the "TO" instruction and other IT operations. For the "TO," the Bit Address register is loaded and the Storage Controller is notified so that any required shifting can be begun. Detailed specification of the other Type 1 operations has been left until the logic design of the IT is done.

The handling of parameters (specified by the P field of the microinstruction) is done directly by combinational logic connected to the P register. The purpose of most P options is apparent. The index insertion codes are used to provide the Match Memory Address field for those word logic functions which require it. For access to a single bit, the lower four bits of the Length register are used directly. For "block" transfers of the stack to and from storage, the lowest four bits of the Bit Count register are used (they are incremented during the execution control operations, so each iteration of the loop accesses a new match memory bit).

4.1.2.3 Storage Controller -- As has been discussed, detailed internal design of the Storage Controller has been deferred to allow a choice of storage technology to be made at the same time as a build commitment. In this section then, the terminal characteristics of the Storage Controller are described. In addition, the internal requirements of the Storage Controller are discussed in the context of the baseline CCD technology.

As shown in Figure 56, the Storage Controller is connected to three other system subunits; the Fast I/O Controller, the Iteration Controller, and the Central Signal Distributor. Functionally, it is responsible for shifting the storage to any bit position required by either of the controllers, generating the control signals to perform required operations, and handling any storage maintenance functions (e. g., refresh).

The design of the slave is such that operations cannot occur simultaneously in the Fast I/O and IT Controllers. Thus, the interfaces to these units may be handled in an identical fashion with confidence that no contention will occur.

When the Storage Controller receives a "REQUEST," it compares the bit address received from the requesting unit with an internal counter indicating the index of the next bit available for an operation. If the values match, the Storage Controller returns a "Ready" signal, indicating that the requestor may proceed to operate on the requested bit. The requestor may then return an "OPENABLE" signal which causes the Storage Controller to perform the operation specified by the "type" code and to increment its internal bit counter. If the bit addresses do not match, the Storage Controller performs the necessary block accessing and shifting prior to returning the "Ready" signal.

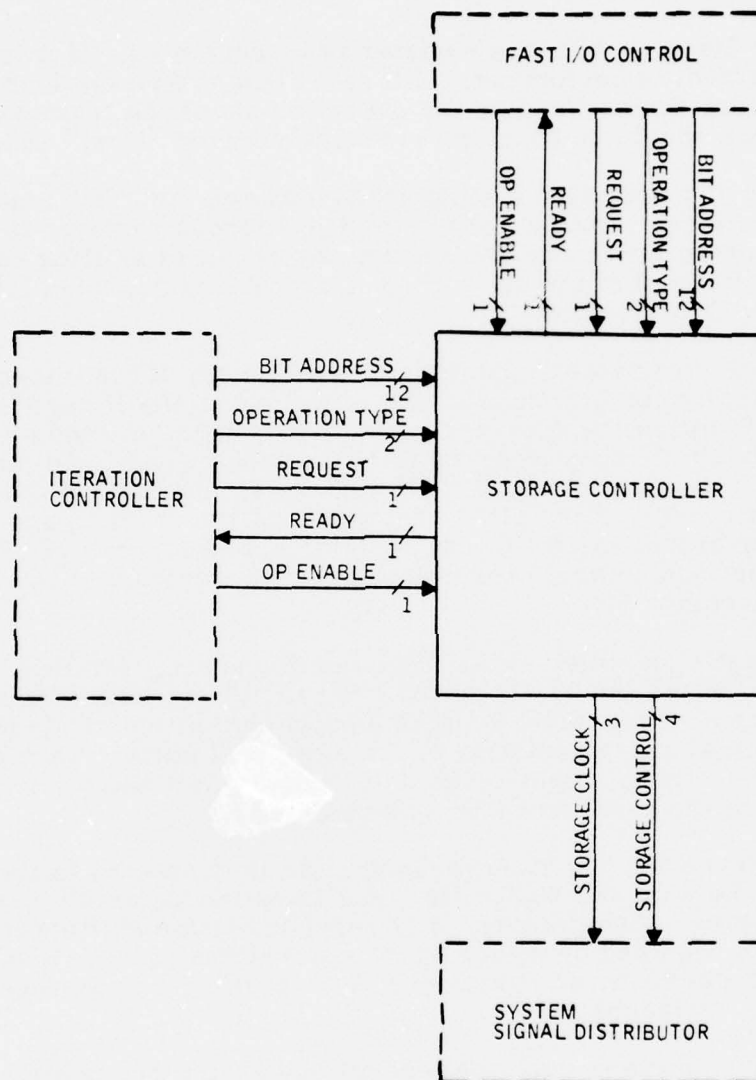


Figure 56. Storage Controller Functions

For baseline purposes, it is estimated that seven lines will be required to control the storage devices. Obviously, this will change with a change in technology.

Internally, a Storage Controller for the baseline CCD part would contain the following registers:

- Bit Position -- A 12-bit register used to hold the current position of the storage. The upper four bits of this register specify the 256-bit block and the lower eight bits the position within the block.
- Block Position Memory -- A 16-word by 8-bit memory containing the current bit position of each of the blocks in a storage word. The lower eight bits of the Bit Position Register may actually be the memory word addressed by the upper four.
- Refresh Timeout -- Some type of timer must be provided so that the Storage Controller can cause a refresh cycle when the time since the last normal array operation has exceeded a threshold.

4.1.2.4 Fast I/O Controller -- It is expected that the Fast I/O Controller will transfer blocks of data between the host's memory and the array without the intervention of the master processor or the host I/O subsystem. The fast I/O operation will be initiated by the Interpreter after the words to participate have been selected (SELECTGROUP). The Fast I/O Controller will use the Storage Controller just as the IT does, transferring the required data and halting.

This capability is provided as a distinct control subunit because the normal host-to-array path has been optimized for control functions and it is expected that the simulations will show that this path is too slow to support a 10^7 bit-per-second rate. If this is not the case, the Fast I/O Controller can easily be deleted from the design.

4.1.3 Host Interface

The interface between the host and the master control unit will physically connect as a high-speed peripheral device of both units. On the host side, this connection can be made via an I/O multiplexer (IOM) on the Contractor's machines or by some equivalent path on other machines. On the master side, connection will be made directly to the main processor bus. From the software viewpoint, transfers will be memory-to-memory.

If the ECAM installation is made with an HIS-6080 as the host, and PDP-11/45 as the master, it is expected that the interface design can be based on a unit designed by Univac, Inc. and installed at the NMIC in Washington, D. C. The unit, called a Bus-to-Bus Converter, is described in Univac specification PX11464. It has an estimated transfer rate capability of 120,000 bytes per second. It is expected that this rate is in balance with the average rate capabilities of other system elements, but would suggest that evaluation of ECAM burst transfer rates be checked via simulation prior to committing to the design.

If another combination of host and master is finally chosen, simulation results can be used to determine the required capability of the interface.

4.2 ECAM ASSOCIATIVE ARRAY

In this section, the functions and logic of the ECAM Associative array are discussed. The discussion covers the word logic and signal distribution system and, briefly, the storage part requirements.

The ECAM Associative array is a memory which has a data processing capability at each individual word. This capability is provided by the Word Logic Blocks (WLBs) which are connected as shown in Figure 57. A WLB operates in a bit serial manner using a field (L) of its respective memory word as a data source or destination. Each block contains an activity control called a match bit (m).

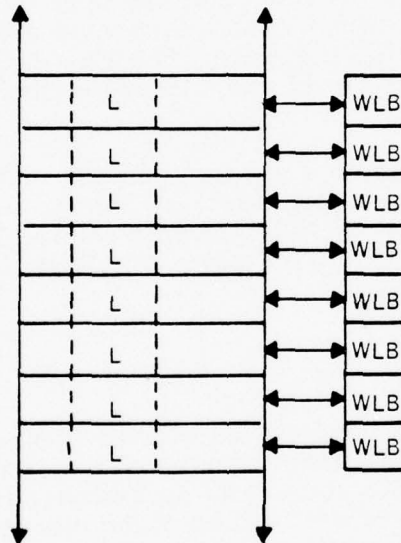


Figure 57. Memory and Word Logic Blocks

The match bit determines the operational state of a WLB. If the match bit is set ($m = 1$), the block is a participant. If the match bit is reset ($m = 0$), the block is a nonparticipant. All WLBs operate in synchronism to execute commands which are broadcast to the array.

Several types of processing are possible. Data processing functions transform data fields by addition or subtraction of a value. They also search participating words to locate the maximum or minimum. A compare operation identifies which are larger, smaller, or equal to a broadcast value. Match manipulation functions allow search and compare results to be

combined. Thus, the array can execute complex search expressions. Input/output functions allow data fields to be entered into or removed from the array words.

To facilitate rapid input and output, the array includes I/O switches which selectively connect the WLBs to 10 parallel I/O lines. Each I/O switch serves 10 WLBs, as shown in Figure 58. During an I/O operation, each I/O switch examines both the WLBs it serves and control information describing the status of the I/O lines. The switch then connects as many as possible of its participating blocks' unused I/O lines. The switch also reports the number of connections made to the I/O control mechanism.

Control signals for the I/O switches are created in the Control Generator Logic (CGL) tree (Figure 59). The tree sums the number of responding WLBs in a hierarchical fashion and produces a list of partial sums which control the I/O switches. The combined logic of the I/O switches and the CGL tree facilitates multiple match resolution and responder counting within the array. The tree also provides a pointer to the physical location of the first responding WLB.

4.2.1 Array Physical Overview

As will be discussed in Section 6, the array is implemented in cabinets of 8.4×10^7 bits or approximately 10 million bytes. Each cabinet contains the following equipment:

- Thirty-two storage/word-logic circuit boards
- Four storage buffer boards (board Type A) cabinet buffer circuit board (board Type B)
- A back panel, cable interface, and external connector assembly
- Modular power supplies
- A cooling system consisting of fans for the power supplies, and fans/ducting for the circuit boards

Control signal distribution throughout the ECAM array is in the form of a fan-out tree as shown in Figure 60. This technique takes advantage of the physical parallel structure of the array. Commands (called function codes) are issued by the slave controller to all words of the array. Distribution of the function code signals is performed by a hierarchy of signal buffers as shown in the figure. Tree-structured logic for I/O control and multiple match resolution is packaged with the signal distribution buffers.

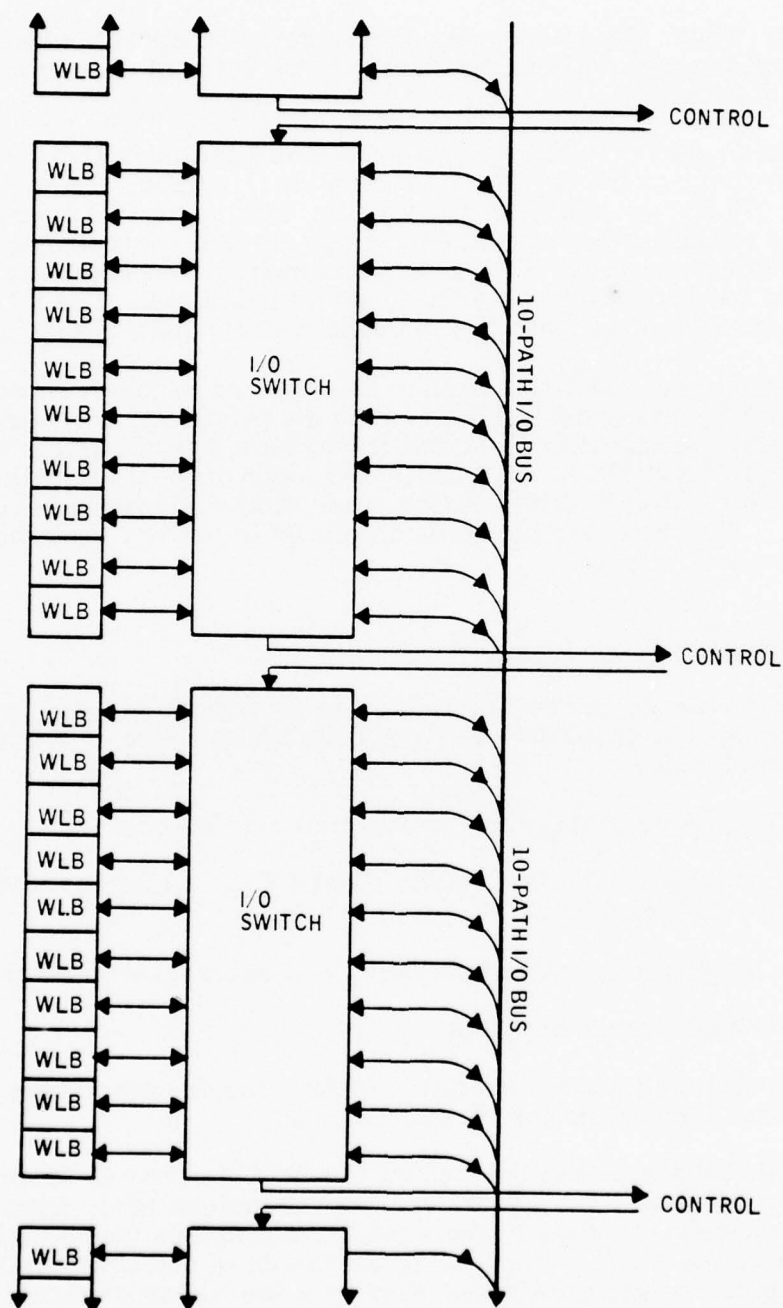
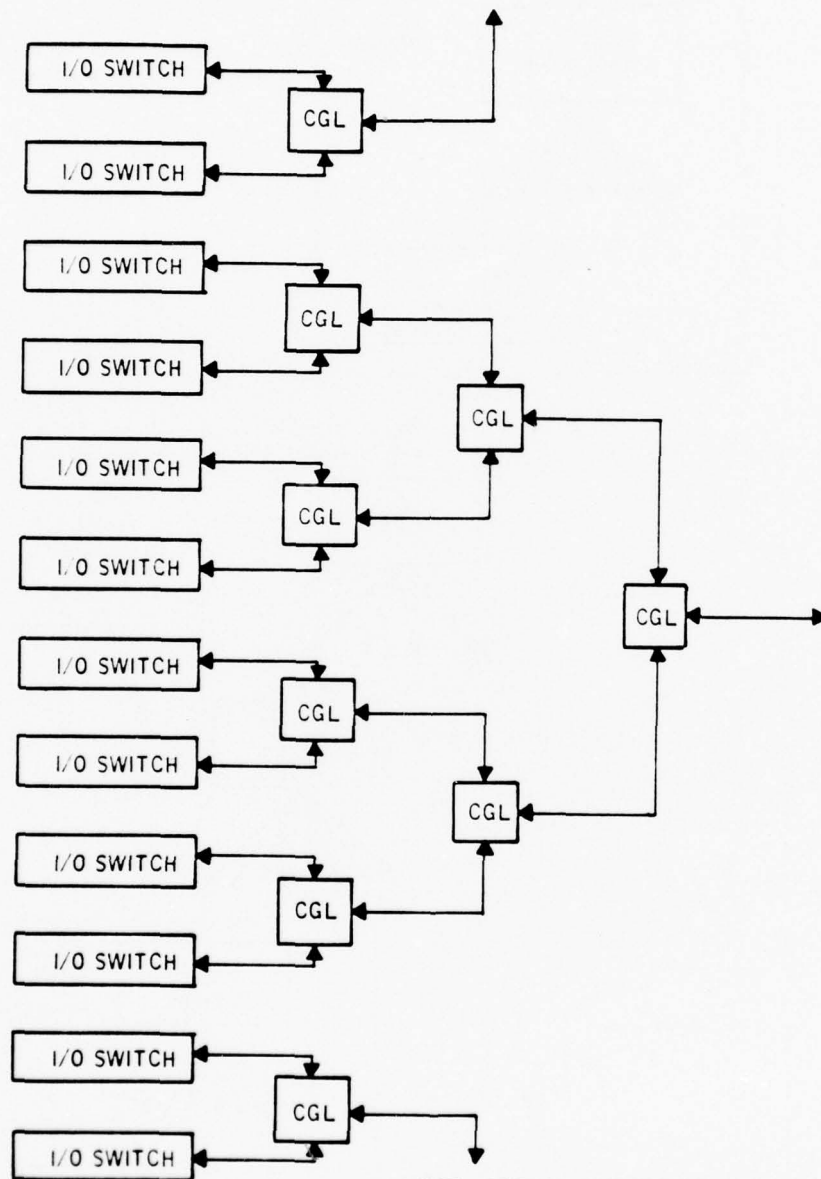


Figure 58. I/O Switch Data Connections



NOTE: ACTUAL IMPLEMENTATION
DIFFERS SLIGHTLY

Figure 59. I/O Switch Control Connections

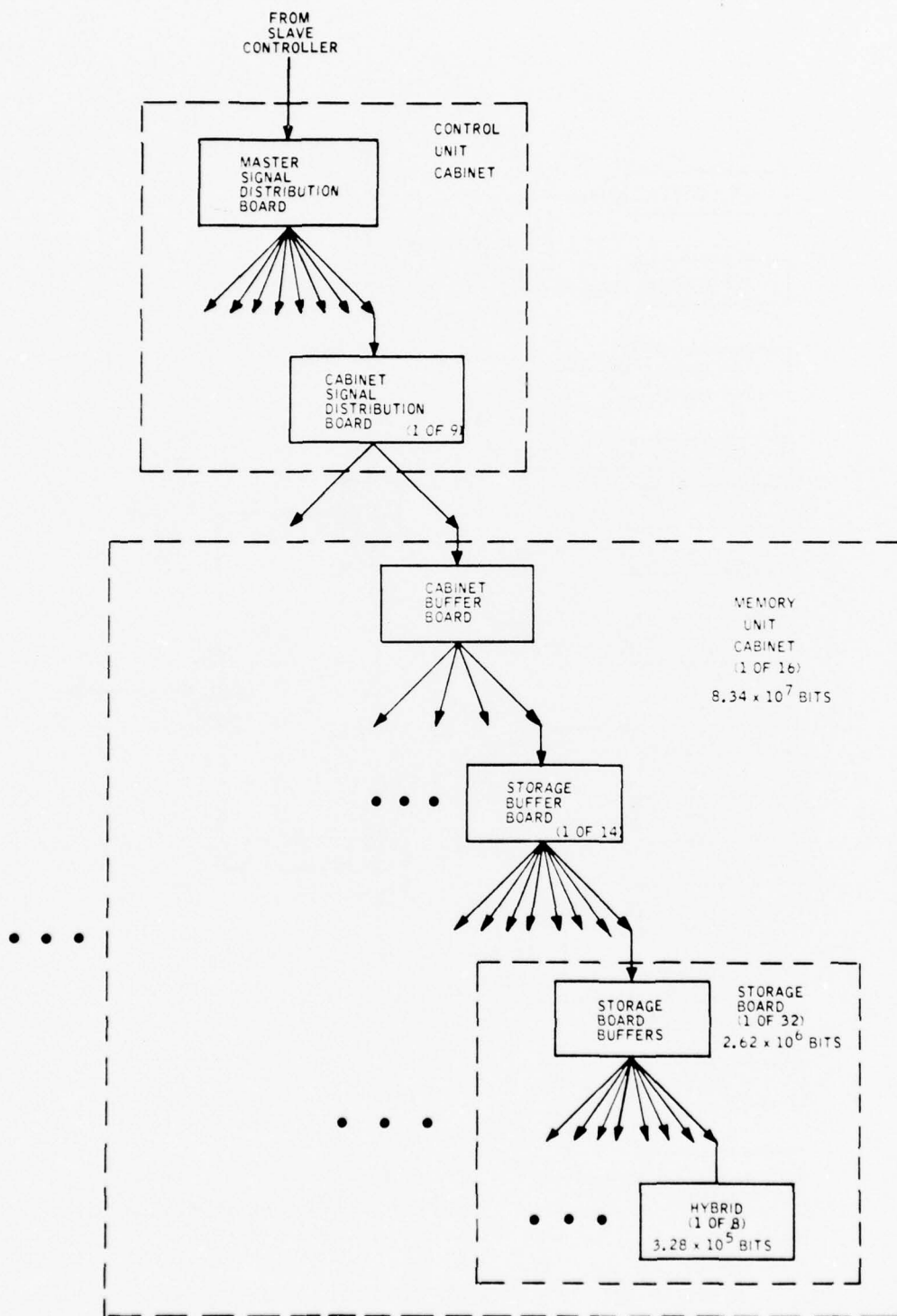


Figure 60. ECAM Array Signal Distribution

A circuit board is the smallest plug-in replaceable unit. It has a storage capacity of over 2.62 million bits. A schematic diagram of one circuit board is shown in Figure 61. Each board contains:

- Eight hybrid circuit modules (all identical)
- Signal distribution buffering circuitry
- Control generator logic (CGL)

The CGL is a tree-structured control circuit implemented in off-the-shelf integrated circuits. It provides the control mechanism for the following functions:

- Input/output
- Multiple match resolution
- Match counting
- Physical address determination of first responding hybrid

To achieve desirable I/O rates, the array has been provided with 10 parallel I/O paths. The CGL allows selection of the array words which are to be connected to the I/O paths and controls the I/O switches (discussed later) which make the actual connection. The circuit board as shown in Figure 61 contains approximately 52 standard integrated circuits and requires approximately 55 connector pins.

Each hybrid circuit module (Figure 62) has a storage capacity of 327.68 Kbits and contains:

- Memory and word logic chips
- Signal distribution buffering circuitry (chips)

To simplify the structure and construction of the hybrid circuit module, the CGL at this level has been designed into the Word Logic LSI Chips (WLLC).

As stated previously the array design has intentionally been made as storage technology-independent as possible. The baseline storage chip organization is 10 words of 4096 bits each. Alternative organizations are discussed in Section 6.

Each of the eight WLLCs in a hybrid consists of:

- Ten WLBs
- A 16-word by 10-bit static RAM (Match Memory)
- A 10-word, bit-serial I/O switch
- I/O and Multiple Match Resolver control logic
- Function Code decoding logic

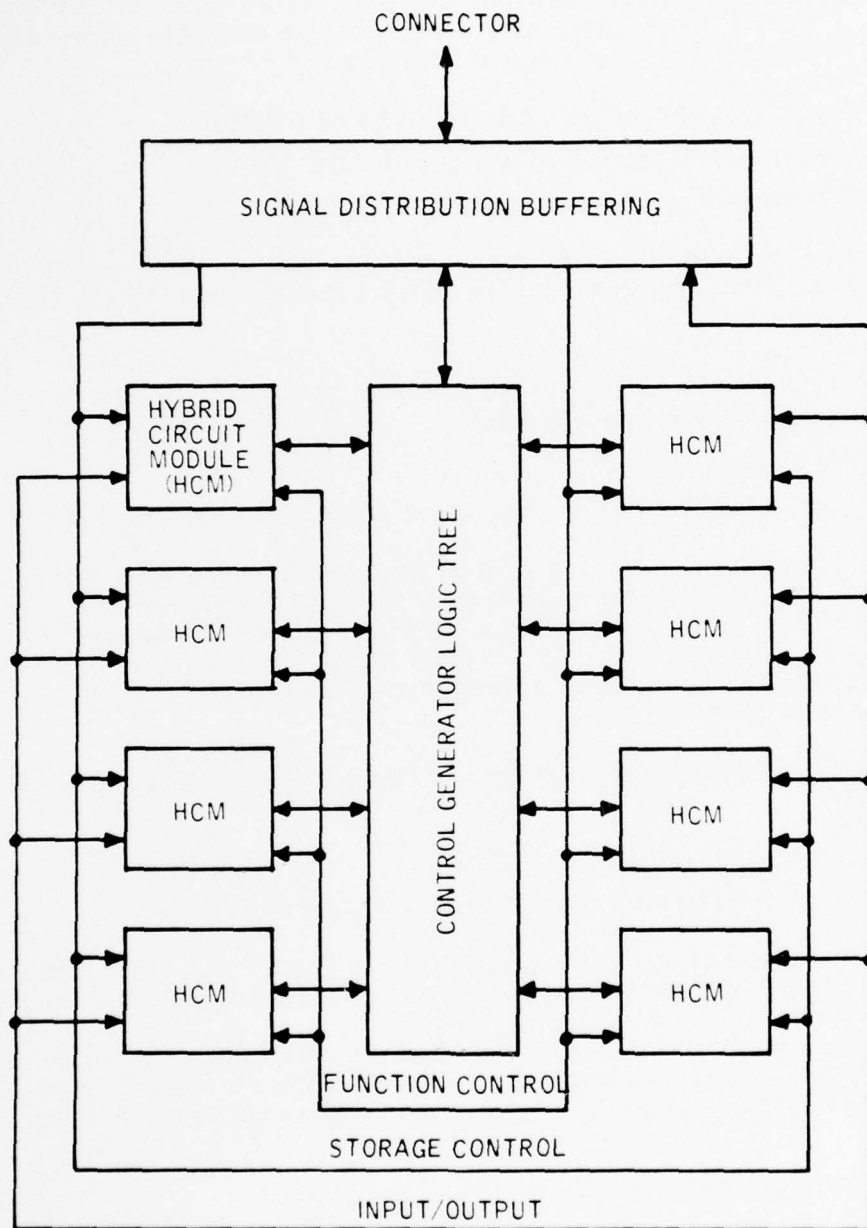


Figure 61. ECAM Storage Board Function Block Diagram

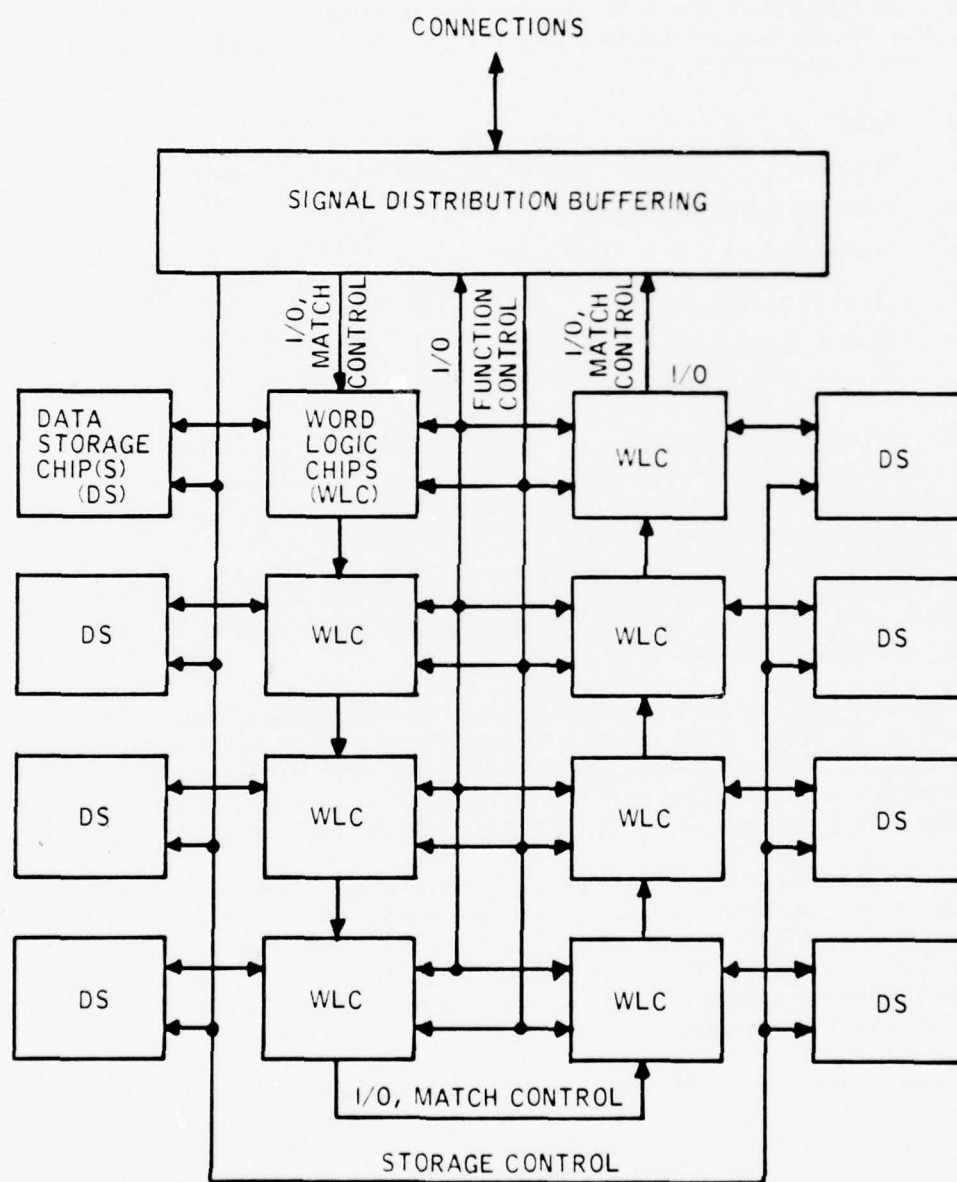


Figure 62. ECAM Hybrid Circuit Module Function Block Diagram

The interconnection of these components is shown in Figure 63.

The WLBs comprise the data processing portion of the ECAM array. Each WLB is directly connected to one storage word (4096 bits). A WLB can perform such functions as:

- Add
- Subtract
- Compare with value
- Input/output
- Maximum search
- Minimum search
- Between limits search

Each WLB (Figure 64) contains a match bit (m). In general, the match bits serve two purposes. Prior to an operation, the match bits indicate those WLBs (and their respective storage words) which will participate in a function. After an operation, the match bits indicate those WLBs (and storage words) which "passed" the function. For example, prior to a Maximum Search function, the match bits indicate the storage words which will be searched; and after the search, the match bits indicate the word (or words) with the maximum value.

In addition to its match bit (m), each array word has a match bit storage memory of 16 bits. This feature facilitates evaluation of complex search expressions by allowing intermediate search results to be stored and later retrieved for combination with the results of additional tests. Word Logic functions are provided for manipulation and storage of match bit results.

To ease implementation of the WLLC, the match bit storage vectors of the 10 WLBs are combined into a single 16-word by 10-bit RAM called the Match Memory (MM). This is shown in Figure 65. Transceivers are provided to reduce the number of interconnections required on the chip. Functionally, the Match Memory still operates as if each WLB had its own dedicated 16-bit match bit storage vector.

The I/O switch (Figure 66) allows the simultaneous bit-serial input or output of up to 10 data storage words. Each WLLC contains one 10-path I/O switch which is connected to the 10 WLBs within the chip. The data I/O lines from all I/O switches are combined (10 parallel) by a hierarchy of logic similar to the signal distribution buffering. The I/O switch is also an integral part of the Multiple Match Resolver control logic within the ECAM array. In single mode, the I/O switch selects the first responding WLB ($m = 1$) within the entire array. In multiple mode, the I/O switch selects the first 10 (or less) responding WLBs. In either case, the respective storage word (words) may then be input or output.

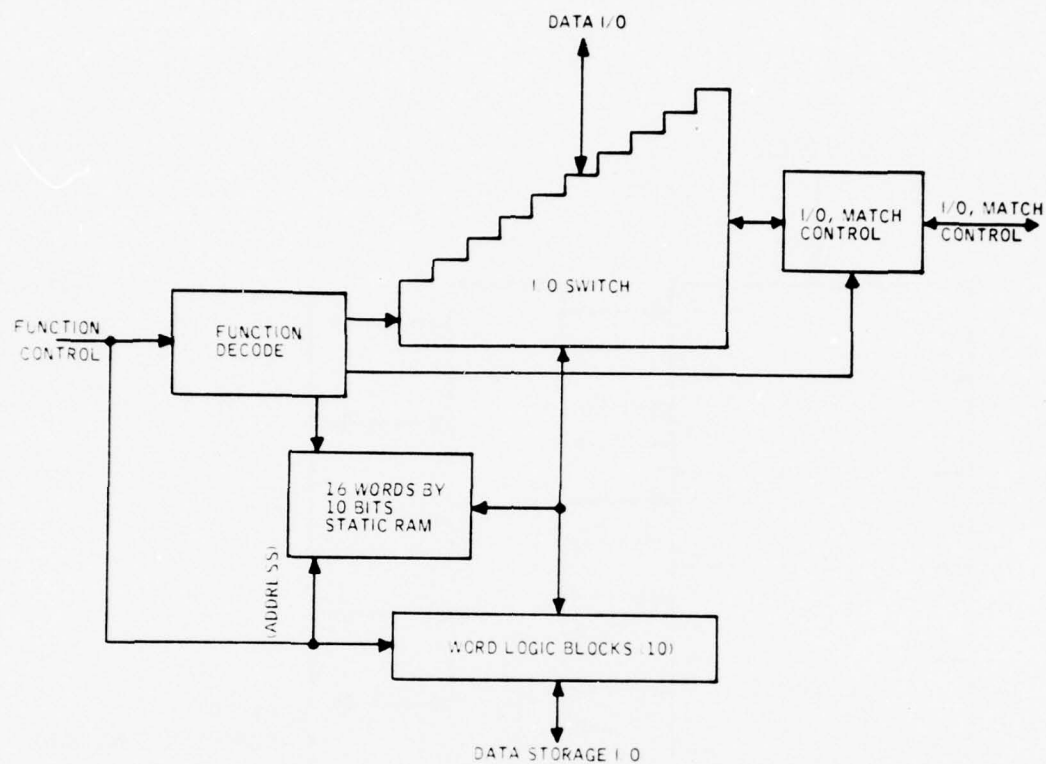


Figure 63. ECAM Word Logic LSI Chip Function Block Diagram

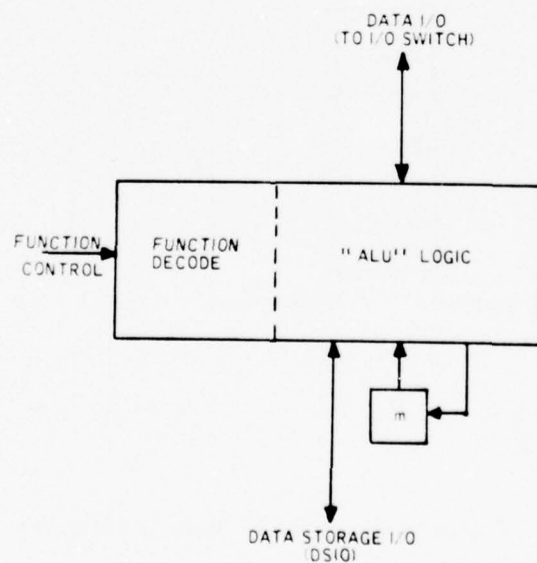


Figure 64. ECAM Word Logic Block

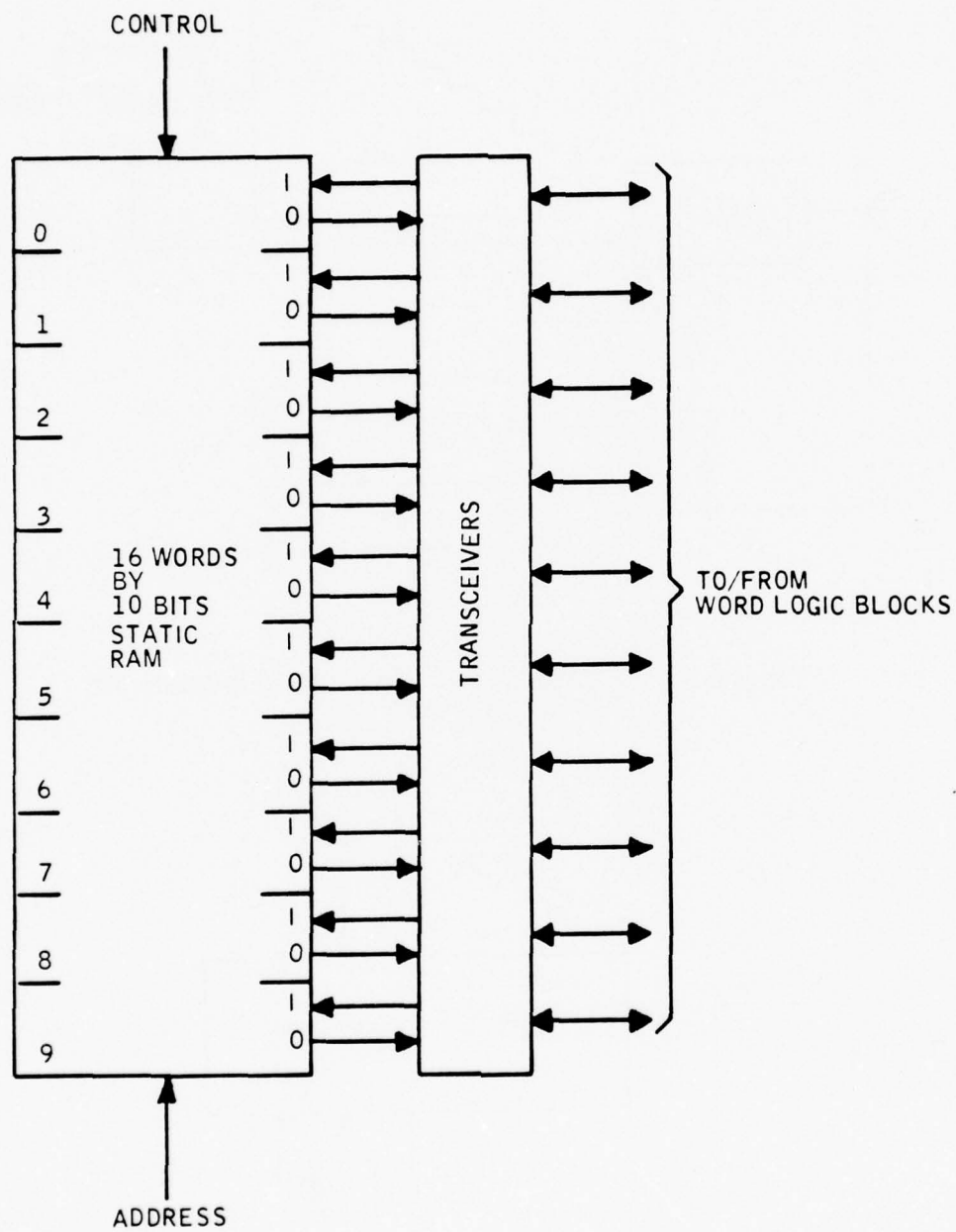


Figure 65. ECAM Match Memory

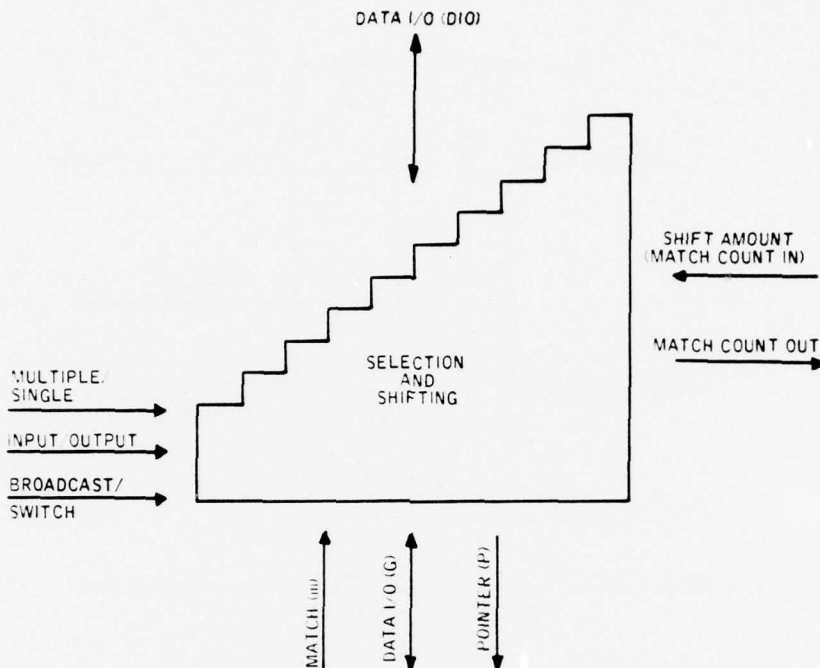


Figure 66. ECAM I/O Switch

To perform the Multiple Match Resolution function, each I/O switch requires control signals which describe the number of detected responding words. This control information is in the form of a list of partial sums. Each partial sum indicates (modulo 16 with overflow) the count of responders beginning at one end of the array. Rather than allow the sum to ripple through the array (a time-prohibitive procedure), a summing tree is provided which computes partial sums via a hierarchy of adders. As mentioned earlier, this logic, called the Control Generator Logic (CGL) tree, provides the control mechanism for:

- Input/output
- Multiple match resolution
- Match counting
- Physical address determination of first responding hybrid

The I/O and Multiple Match Resolver control logic (Figure 67) generates and combines match resolver signals in conjunction with the I/O switch and the CGL tree.

The functions of the WLLC are directed by a six-bit Function Code (FC). The FC is decoded for I/O switch and Match Memory control by the FC decoding logic. Also, each WLB contains logic (all identical) to decode the FC lines.

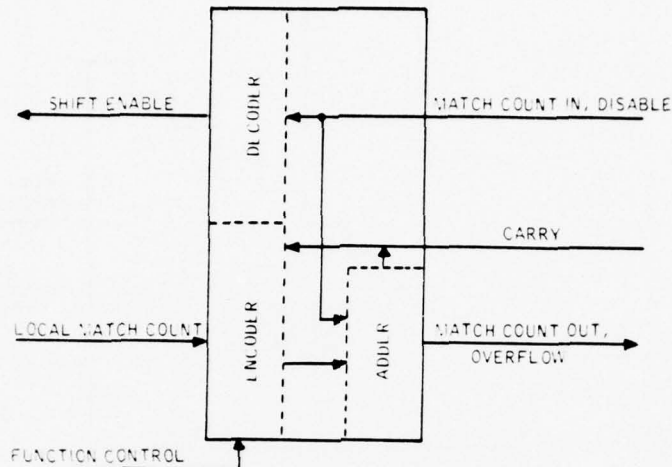


Figure 67. I/O and Multiple Match Resolver Control Logic

4.2.2 Word Logic LSI Chip (WLLC)

This chip (shown in Figure 68) is the bottom level of the ECAM Associative array structure. It includes five functional subunits: The WLB, Match Memory, I/O switch, I/O and MMR control logic, and the Function Decode logic. These subunits are discussed below.

4.2.2.1 Word Logic Block -- This section discusses both the hardware of the WLB and the processing functions which it performs. For completeness, the I/O and MMR functions which affect the WLB are included in this section also.

The WLB (Figure 69) is the data processing element of the ECAM array. Three flip-flops determine the state of the block:

- Storage Latch (L)
- Match Bit (m)
- Temporary (T)

The Storage Latch captures and holds the most recent (current) data bit read out of the Data Storage chip. The Match Bit is used to indicate that the block is an active participant in the current operation (search, count, I/O, etc.). The temporary bit serves several purposes, including Data Storage, Status Indication, and Control.

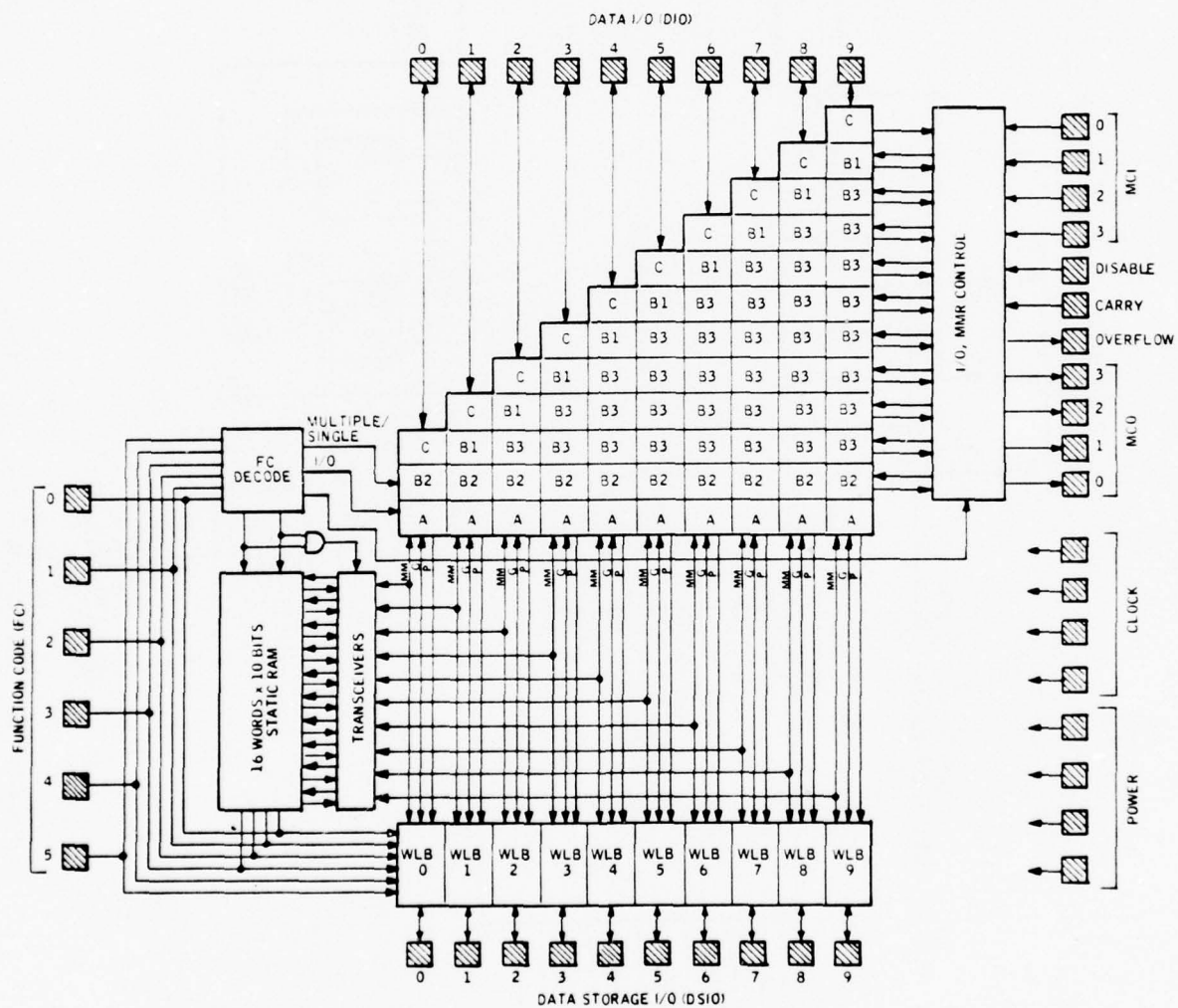


Figure 68. Word Logic LSI Chip Function Block Diagram

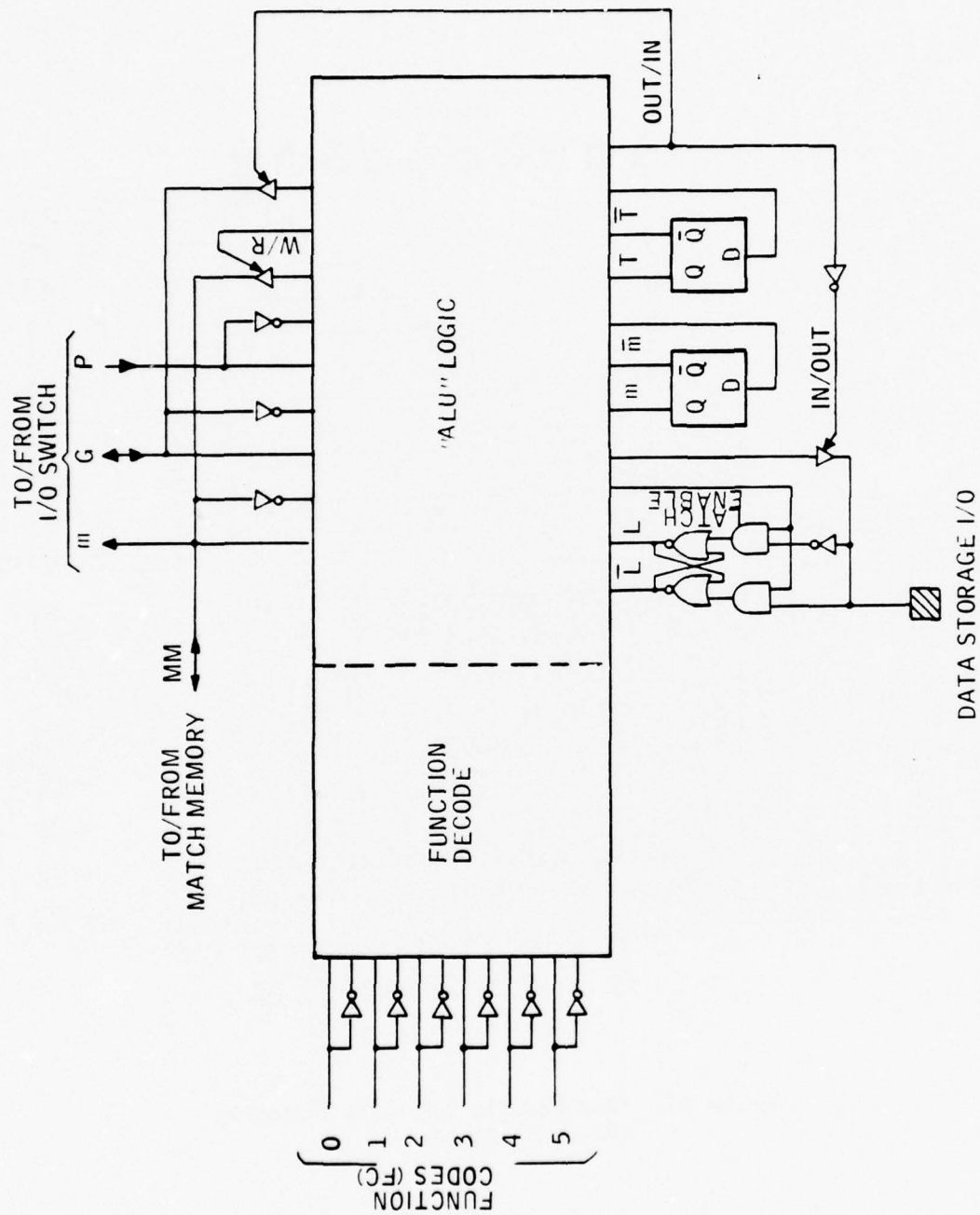


Figure 69. ECAM Word Logic Block

Processing of both data and state is performed by a combinatorial logic block. The operation of the block may be conceptually broken into two steps:

- Decode
- Processing

One part of the block decodes the six function control lines from the slave controller. The decoded function opcodes direct data and state processing which takes place in the "ALU" portion of the block. Implementation of the combinatorial logic block will be in the form of a programmable logic array (PLA) or read-only memory (ROM).

The combinatorial logic block also provides outputs to (and accepts inputs from) the Match Memory (MM) and the I/O switch. The match bit may be stored in (or loaded from) the Match Memory. The match bit line also supplies the I/O switch with information necessary for multiple match resolution and high-speed I/O. The Global Data line (G) transfers data to and from the I/O switch. The Pointer (P) is an output from the Multiple Match Resolver circuitry that indicates the selected (resolved) word or words.

The ECAM word logic can perform three types of processing functions:

- Match manipulation
- Data processing
- Input/output

In general, these functions are supplied by the decoding of function codes (FCs) and the combination of the decodes with array state or data.

In the function descriptions which follow, state or data variables which do not have a new value explicitly stated remain unchanged. Also, some functions may require setup processing (usually state manipulation) prior to the iteration of the function across bits of the selected data field.

4.2.2.1.1 Match Manipulation Functions -- The Match Manipulation functions (Table 46) provide the capability of saving, restoring, manipulating, and initializing word logic state contained in the match bit (m) and the temporary bit (T). The Match Memory functions move match state to or from the Match Memory. The Boolean functions perform logical operations on m and T. The Initialize functions load m and T from the Global Data line (G). The Input/Output and Multiple Match Resolver functions determine the operational state of the I/O switch and Multiple Match Resolver as well as effecting match resolution.

Table 46. Match Manipulation Functions

Function	Operation
Match Memory	<ul style="list-style-type: none"> • Match Memory to Match Bit ($m \leftarrow MM_1$) • Match Memory to Storage ($L \leftarrow MM_1$) • Storage to Match Memory ($MM_1 \leftarrow L$) • Match Bit to Match Memory ($MM_1 \leftarrow m$)
Boolean	<ul style="list-style-type: none"> • AND ($m \leftarrow m \cdot T$) • OR ($m \leftarrow m + T$) • Exclusive OR ($m \leftarrow m \oplus T$) • Complement m ($m \leftarrow \bar{m}$) • Exchange ($m \leftrightarrow T$) • Copy ($T \leftarrow m$) • Complement T ($T \leftarrow \bar{T}$)
Initialize	<ul style="list-style-type: none"> • Initialize m ($m \leftarrow G$) • Initialize T ($T \leftarrow G$)
I/O and MMR Control	<ul style="list-style-type: none"> • Zero Hybrid Interface Register • Load Hybrid Interface Register • Capture Carry • Report Carry • Report Matches
Multiple Match Resolver	<ul style="list-style-type: none"> • Select First $m \leftarrow P$ (slow) • Select First Set $m \leftarrow P$ (fast) • Discard First $m \leftarrow m \cdot \bar{P}$ (slow) • Discard First Set $m \leftarrow m \cdot \bar{P}$ (fast)

4.2.2.1.1.1 Match Memory Functions -- The Match Memory functions move match state information to or from the Match Memory. Transfers to or from the match bit (m) and to or from the data storage (L) are possible. The temporary bit (T) is used as a control variable. Addressing of the Match Memory is performed by the four least significant function control lines ($FC_{3,2,1,0}$). The possible match state information transfers are shown in Figure 70. The Match Memory functions are:

- 1) Match Memory to Match Bit. The Match Memory to Match Bit function transfers the value of the addressed Match Memory bit (MM_i) of each array word to the match bit (m) of each array word. All words participate in the transfer:

$$m \leftarrow MM_i$$

The temporary bit (T) must be initialized to 1 ($T \leftarrow 1$). The transfer equation is:

$$m \leftarrow (T \cdot MM_i) + (\bar{T} \cdot m)$$

This equation is also used to implement the Match Memory to Storage function when the temporary bit equals zero.

- 2) Match Memory to Storage. The Match Memory to Storage function transfers the value of the addressed Match Memory bit (MM_i) of the participating array words ($m = 1$) to the current bit position of the selected data storage field (L). The storage field of non-participant words remains unchanged. The transfer is:

$$L \leftarrow MM_i$$

The temporary bit (T) must be initialized to zero ($T \leftarrow 0$). The function may be repeated to move all or part of the Match Memory contents. The iteration equations are:

$$L \leftarrow (m \cdot MM_i) + (\bar{m} \cdot L)$$

$$m \leftarrow (T \cdot MM_i) + (\bar{T} \cdot m)$$

These equations are also used to implement the Match Memory to Match Bit function when the temporary bit equals 1.

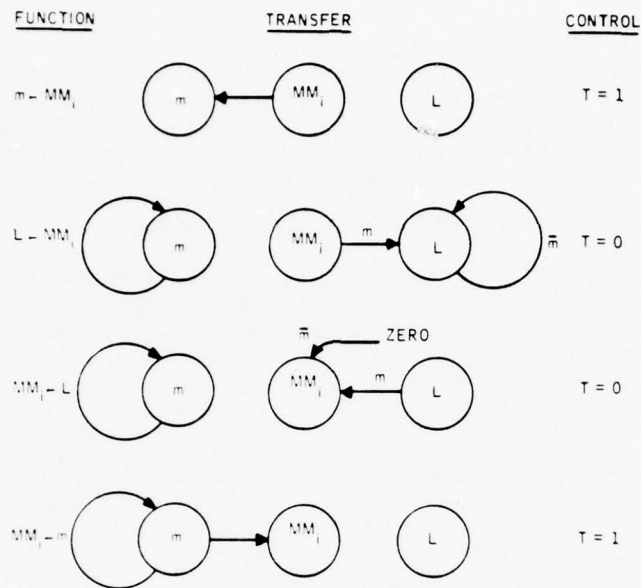


Figure 70. Match State Information Transfers

3) Storage to Match Memory. The Storage to Match Memory function transfers the value of the current bit position of the selected data storage field (L) of the participating array words ($m = 1$) to the addressed Match Memory bit (MM_i). The Match Memory bit of non-participant words is cleared (set to zero). The transfer is:

$$MM_i \leftarrow L$$

The temporary bit (T) must be initialized to zero ($T = 0$). The function may be repeated to load all or part of the Match Memory. The iteration equation is:

$$MM_i \leftarrow m \cdot (L + T)$$

The equation is also used to implement the Match Bit to Match Memory function when the temporary bit equals 1.

4) Match Bit to Match Memory. The Match Bit to Match Memory function transfers the value of the match bit (m) of each array word to the addressed Match Memory bit (MM_i) of each array word. All words participate in the transfer:

$$MM_i \leftarrow m$$

The temporary bit (T) must be initialized to 1 ($T = 1$). The transfer equation is:

$$MM_i = m(L + T)$$

The equation is also used to implement the Storage to Match Memory function when the temporary bit equals zero.

4.2.2.1.1.2 Boolean Functions -- The Boolean functions perform logical operations on the match bit (m) and the temporary bit (T). All array words participate and no initialization is required:

1) AND. The AND function replaces the match bit (m) with the logical product of m and the temporary bit (T):

$$m = m \cdot T$$

2) OR. The OR function replaces the match bit (m) with the logical sum of m and the temporary bit (T):

$$m = m + T$$

3) Exclusive OR. The Exclusive OR function replaces the match bit (m) with the Exclusive OR of m and the temporary bit (T):

$$m = m \oplus T$$

4) Complement m. The Complement m function replaces the match bit (m) with its logical complement:

$$m = \bar{m}$$

5) Exchange. The Exchange function replaces the match bit (m) with the value of the temporary bit (T) and simultaneously replaces T with the value of m:

$$m \leftrightarrow T$$

6) Copy. The Copy function replaces the temporary bit (T) with the value of the match bit (m):

$$T = m$$

7) Complement T. The Complement T function replaces the temporary bit (T) with its logical complement:

$$T = \bar{T}$$

4.2.2.1.1.3 Initialize Functions -- The Initialize functions transfer the value supplied by the slave controller via the Global Data line (G) to either the match bit (m) or the temporary bit (T). All array words participate:

- 1) Initialize m. The Initialize m function replaces the match bit (m) with the value supplied by the slave controller via the Global Data line (G):

$$m \leftarrow G$$

- 2) Initialize T. The Initialize T function replaces the temporary bit (T) with the value supplied by the slave controller via the Global Data line (G):

$$T \leftarrow G$$

4.2.2.1.1.4 I/O and MMR Control Functions -- The Input/Output (I/O) and Multiple Match Resolver (MMR) control functions determine the operational state of the CGL tree, and consequently the state of the I/O switch and Multiple Match Resolver. Proper operational state is required for:

- Input/output
- Multiple match resolution
- Responder counting
- Responding hybrid address determination

Input/Output Control Functions Description:

- 1) Zero Hybrid Interface Register. The Zero Hybrid Interface Register function clears all Hybrid Interface registers as the first step in establishing the proper operational state of the CGL tree:

$$HIF \leftarrow 0$$

- 2) Load Hybrid Interface Register. The Load Hybrid Interface Register function causes all Hybrid Interface registers to capture and retain their respective code values as computed by the CGL tree. The captured values are applied to the hybrid circuit modules as the Match Count In. This function is the second step in I/O switch control preparation and multiple match resolution:

$$HIF \leftarrow \text{Count}$$

3) Capture Carry. The Capture Carry function causes all WLLCs to capture and retain their respective carry values as computed by the CGL tree and by the I/O, MMR control logic on each chip. This function is one of several which constitute an iteration process for counting responders.

4) Report Carry. The Report Carry function causes all WLLCs to present their carry value (1 or zero) to the match count adders. This function is one of several which constitute an iteration process for counting responders. Under normal (noncounting) conditions, a match count value generated by the I/O switch is presented to the adders. The WLLCs are restored to their normal state by the Report Matches function.

5) Report Matches. The Report Matches function causes all WLLCs to present their I/O switch-generated Match Count value to the match count adders. This is the operational state required for I/O switch control preparation, multiple match resolution, and responding hybrid address determination.

Multiple Match Resolver Functions Description: The Multiple Match Resolver functions modify the vector of match bits (m) across the array in accordance with the result of the Multiple Match Resolver logic. The resolver produces a pointer vector (P) which may indicate either the first true match bit ($m = 1$) or the first 10 or less true match bits. The ordering implied by the word "first" is determined by the wiring of the resolver. This ordering is arbitrary, but deterministic. The CGL tree must be in the proper operational state before the Multiple Match Resolver functions will produce valid results. The Multiple Match Resolver functions are:

1) Select First. The Select First function resets all match bits (m) except the first one as indicated by the pointer vector (P):

$$m \leftarrow P \text{ (slow)}$$

2) Select First Set. The Select First Set function resets all match bits (m) except the first 10 as indicated by the pointer vector (P). If less than 10 match bits are true prior to the execution of the function, then no bits will be reset:

$$m \leftarrow P \text{ (fast)}$$

3) Discard First. The Discard First function resets the first true match bit (m) as indicated by the pointer vector (P):

$$m \leftarrow m \cdot \bar{P} \text{ (slow)}$$

4) Discard First Set. The Discard First Set function resets the first 10 true match bits (m) as indicated by the pointer vector (P). If less than 10 match bits are true prior to the execution of the function, then all bits will be reset:

$$m \leftarrow m \cdot \bar{P} \text{ (fast)}$$

4.2.2.1.2 Data Processing Functions -- The data processing functions are: Add/Subtract, Reverse Subtract, Arithmetic Compare, and Minimum/Maximum. They are arithmetic and relational operations that are performed over fields contained within participating words of the ECAM array. The result of these functions is either a transformed data field or a match result. If the word logic match bits (m) are viewed as a vector indicating the word participants of a data processing function, then a match result may be viewed as a transformation of the match vector so that the resultant match vector indicates those words which were both participants and also satisfied the condition evaluated by the function.

4.2.2.1.2.1 Add/Subtract -- The Add/Subtract function is a multiple-use function which adds the global value (G) supplied by the slave controller to the value in the selected data field of each participating word ($m = 1$) and replaces the data field with the result of the addition/subtraction. The operation is bit-serial beginning with the LSB. The currently addressed data storage bit is held by the data storage latch (L) within the WLB.

The transformations which are possible with the Add/Subtract function include:

$$\begin{aligned} L &\leftarrow L + G \\ L &\leftarrow L - G \\ L &\leftarrow L + 1 \\ L &\leftarrow L - 1 \\ L &\leftarrow L + 0 \\ T &\leftarrow G \leq L \end{aligned}$$

For each data bit, the iteration proceeds by placing the binary sum of L, G, and T into L. Simultaneously, the binary carry is placed in T. At the completion of the iteration process, T contains the carry-out of the MSB.

The iteration equations are:

$$\begin{aligned} L &\leftarrow m (L \oplus G \oplus T) + \bar{m} T \\ T &\leftarrow m (L \cdot G + L \cdot T + G \cdot T) + \bar{m} T \end{aligned}$$

Note that the values of L and T remain unchanged in those words which are nonparticipants ($m = 0$), and that the match bit of all WLBs remains unchanged.

Details of the Add/Subtract function transformations are as follows:

- 1) $L \leftarrow L + G$. This use of the Add/Subtract function requires that T be initialized to zero for all participating words (state manipulation). The global value (G) is then supplied by the slave controller, LSB first, one bit at a time (one bit for each iteration of the function).
- 2) $L \leftarrow L - G$. To perform subtraction of the global value, the temporary storage bit (T) of each participating word must be initialized to 1 (carry-in = 1). Then, the global value (G) is supplied in one's complement form. Alternatively, T may be initialized to zero and G then supplied as two's complement.
- 3) $L \leftarrow L + 1$. Incrementing a data field by 1 is done by initializing T to 1 ($T \leftarrow 1$) and adding zero ($G = 0$).
- 4) $L \leftarrow L - 1$. Decrementing a data field by 1 is done by initializing T to zero ($T \leftarrow 0$) and adding a global value of all ones (two's complement form of minus 1).
- 5) $L \leftarrow L + 0$. The Add/Subtract function will result in no change to the data field if T is initialized to zero ($T \leftarrow 0$) and the global value is also zero ($G = 0$).
- 6) $T \leftarrow G \leq L$. This comparison may be performed LSB first by executing the Add/Subtract function in its $L \leftarrow L - G$ form and noting the value of T at completion. A value of 1 ($T = 1$) indicates that the global value was less than or equal to the field value. Both values are assumed to be unsigned integers. Normally, T should be moved to m (by $m \leftarrow m + T$) to record the result of the comparison.

4.2.2.1.2.2 Reverse Subtract--The Reverse Subtract function subtracts the value of the selected data field (L) of each participating word ($m = 1$) from the global value (G) supplied by the slave controller, and replaces the data field with the result of the subtraction. The data transformation may thus be described as:

$$L \leftarrow G - L$$

The operation is bit-serial beginning with the LSB. The temporary bit (T) must be initialized to 1 ($T \leftarrow 1$) prior to the iteration of the Reverse Subtract function.

For each data bit, the iteration proceeds by placing the binary sum of \bar{L} , G, and T into L. Simultaneously, the binary carry (borrow) is placed in T. At the completion of the iteration process, T contains the borrow generated by the MSB.

The iteration equations are:

$$L \leftarrow m (\bar{L} \oplus G \oplus T) + \bar{m} L$$

$$T \leftarrow m (\bar{L} \cdot G + \bar{L} \cdot T + G \cdot T) + \bar{m} T$$

The values of L and T remain unchanged in those words which are non-participants ($m = 0$), and the m bit of all WLBs remains unchanged.

4.2.2.1.2.3 Arithmetic Compare--The Arithmetic Compare function compares the value of the selected data field (L) of each participating word ($m = 1$) with the global value (G) supplied by the slave controller, and codes the result of the comparison into m and T:

<u>m</u>	<u>T</u>	<u>Condition</u>
0	1	$L < G$
1	1	$L = G$
1	0	$L > G$
0	0	Nonparticipant

Proper operation of this function requires that the temporary bit (T) be made equal to the match bit (m) for all words in the ECAM array (i. e. , $T \leftarrow M$) prior to beginning the iterations. The operation is bit-serial beginning with the MSB.

For each data bit, the iteration proceeds by comparing L and G and resetting the match bit (m) if $L < G$, or resetting the temporary bit (T) if $L > G$. Once either bit is reset in a particular WLB, neither bit will be changed again (only one reset can occur per word). At the end of the iteration process, WLBs which have both m and T still true correspond to words where $L = G$.

The iteration equations are:

$$m \leftarrow m (\bar{T} + \bar{G} + L)$$

$$T \leftarrow T (\bar{m} + G + \bar{L})$$

This function does not change the contents of the data storage. Normally, the encoded result should be moved to the match bit (m) or the Match Memory (MM) in order to prevent the value of T from being lost during subsequent processing.

4.2.2.1.2.4 Minimum/Maximum -- The Minimum/Maximum function is a dual-use function which selects the participating word ($m = 1$) which contains the minimum (or maximum) value data field (L). The selected word is

indicated by a match bit value of 1. (The match bits of unselected participants are reset.) The two possible transformations are:

$$m \leftarrow m \cdot (\text{minimum } L)$$

or

$$m \leftarrow m \cdot (\text{maximum } L)$$

Multiple responses will result when more than one word contains the minimum (or maximum) value. The operation is bit-serial beginning with the MSB.

The iteration process consists of two steps per bit of the selected data field. The first step is the application of the Output function (Section 4.2.2.1.3). In the second step the slave controller broadcasts the value produced by the Output function back to each word for comparison. The Output function in this application results in the OR of the current bit from all participating words. The iteration equations are:

$$G \leftarrow m \cdot (T \oplus L) < \text{output ORing} >$$

and

$$m \leftarrow m \cdot [T(\bar{L} + \bar{G}) + \bar{T}(L + G)]$$

The value of L is the same for both steps of the iteration.

Details of the Minimum/Maximum function are as follows:

- 1) Minimum. This use of the Minimum/Maximum function requires that T be initialized to 1 for all participating words ($T \leftarrow m$). This causes the output function to apply \bar{L} to the output OR. A true value (1) from the output OR indicates that at least one of the participating words contained a zero at the current bit position. If a zero is detected, the Minimum/Maximum function (with $T = 1$) will retain as participants only those participating words which contained a zero. If a zero was not detected, all participants will be retained. At the completion of the iteration process, only the word (or words) containing the minimum value in the selected data field (L) will be a participant.
- 2) Maximum. This use of the Minimum/Maximum function requires that T be initialized to zero for all participating words ($T \leftarrow 0$). This causes the output function to apply L to the output OR. A true value (1) from the output OR indicates that at least one of the participating words contained a 1 at the current bit position. If a 1 is detected, the Minimum/Maximum function (with $T = 0$) will retain as participants only those participating words which contained a 1. If a 1 was not detected, all participants will be retained. At the completion of the iteration process, only the word (or words) containing the maximum value in the selected data field (L) will be a participant.

4.2.2.1.3 Input/Output Functions -- The Input/Output functions are: Input, Output, Flag Duplicates, Broadcast, and Output OR. They allow the loading or unloading of words (or fields within words) in the ECAM array. The I/O switch permits up to 10 array words to be loaded or unloaded simultaneously in a bit-serial manner. Additionally, a single value may be broadcast to all array words, or all words may be combined by logical ORing to form a single output.

Details of the Input/Output functions are as follows:

- 1) Input. The Input function transfers data from the Global Data lines (G) to the selected data field (L) of each participating word ($m = 1$). The data transfer is:

$$L \leftarrow G$$

The operation is bit-serial beginning with either the LSB or MSB. No initialization is required and up to 10 words may be loaded simultaneously.

For each data bit, the iteration replaces the data storage bit with the binary value of one Global Data line. The mapping of Global Data lines to array words is performed by the I/O switch (Section 4.2.2.3). The iteration equation is:

$$L \leftarrow m \cdot G + \bar{m} \cdot L$$

Nonparticipant words ($m = 0$) remain unchanged.

- 2) Output. The Output function transfers data from the selected data field (L) of each participating word ($m = 1$) to the Global Data lines (G). The data transfer may be in either true (L) or complement (\bar{L}) form. Thus, the data transfers which are possible are:

$$G \leftarrow L$$

$$G \leftarrow \bar{L}$$

The operation is bit-serial beginning with either the LSB or the MSB. The temporary bit (T) must be initialized to zero ($T = 0$) for true data transfer, and to 1 ($T = 1$) for complement data transfer. Up to 10 words may be unloaded simultaneously.

For each data bit, the iteration places the value of the data storage bit (true or complement) on the Global data line. The mapping of array words to Global Data lines is performed by the I/O switch (Section 4.2.2.3). The iteration equation is:

$$G \leftarrow m \cdot (T \oplus L)$$

All words, fields, and match bits remain unchanged.

3) Flag Duplicates. The Flag Duplicates function is an adjunct to the Output function. It detects those participating words ($m = 1$) in which the selected data field (L) is equal to the field of the word being output from the array. The state transformation is:

$$T \leftarrow L \oplus G$$

The operation is bit-serial beginning with either the LSB or the MSB. The temporary bit (T) must be initialized to 1 ($T \leftarrow 1$).

For each data bit, the iteration process consists of two steps. First, the Output function is applied. Then, during the Flag Duplicates function, the slave controller broadcasts the output data bit back to the array for comparison. The Global Data line (G) is compared with the current data bit (L) and the result is placed in the temporary bit (T). The iteration equation is:

$$T \leftarrow T \cdot (\bar{L} \oplus G)$$

The match bit (m) of all WLBs remains unchanged.

4) Broadcast. The Broadcast function transfers data from the first Global Data line (G_0) to the selected data field (L) of each participating word ($m = 1$). The data transfer is:

$$L \leftarrow G_0$$

The operation is bit-serial beginning with either the LSB or the MSB. No initialization is required and all array words may be loaded simultaneously with a single value.

For each data bit, the iteration replaces the data storage bit with the binary value of the first Global Data line, G_0 . The I/O switch operates in the Broadcast mode to perform this function. The iteration equation is:

$$L \leftarrow m \cdot G_0 + \bar{m} \cdot L$$

Nonparticipant words ($m = 0$) remain unchanged.

5) Output OR. The Output OR function transfers data from the selected data field (L) of each participating word ($m = 1$) to the first Global Data line (G_0). In the process, the bit output of each array word is logically ORed with the bit outputs of all other participating words. The source data may be in either true (L) or complement (\bar{L}) form. Thus, the data transfers which are possible are:

$$G_0 \leftarrow L \text{ (ORing)}$$

$$G_0 \leftarrow \bar{L} \text{ (ORing)}$$

The operation is bit-serial beginning with either the LSB or the MSB. The temporary bit (T) must be initialized to zero ($T = 0$) for true data ORing, and to 1 ($T = 1$) for complement data ORing. All words of the array may be ORed as output simultaneously.

For each data bit, the iteration OR's the value of the data storage bit (true or complement) onto the first Global Data line. The selection and ORing is performed by the I/O switch which operates in the Output ORing mode to perform this function. The iteration equation is:

$$G_0 \leftarrow m \cdot (T \oplus L)$$

All words, fields, and match bits remain unchanged.

4.2.2.1.4 ECAM Associative Array Functions Summary -- Table 47 presents the ECAM Associative array functions in tabular form. The meanings of the columns are discussed in the following paragraphs.

Control of the array is established by both the word logic FC lines and the storage control lines (shift, RMW). The first 32 FCs (00-1F) use the four LSBs (3, 2, 1, 0) to address the Match Memory. Those same codes also use the temporary flip-flop (T) as an additional control variable. Note that only the Read and Modify modes of the storage device are used. The Write mode is not required. Function Codes 3D through 3F are currently shown as performing no operation (i. e., they are identical to NOP, 37). Those codes are reserved for maintenance functions which have not been defined. One possible maintenance feature would be the inclusion of a Fault flip-flop (F) in each WLB. Reserved codes would be used to control and sense this flip-flop. A short discussion of the use of such a flip-flop is presented in Section 6.4.

The Function column describes each array activity in symbolic form. The Local Match Input column lists the logic equations applied to the input of the match bit flip-flop (M). The Temporary Input column lists similar equations for the temporary flip-flop (T).

The Match Memory requires both input and control. The input equations are listed under the MM_i Input heading. The symbol X indicates a "don't care" input. The equations shown also serve as input to the I/O switch. Therefore, some functions list a value even though the Match Memory is not affected. The Read/Write control is meaningful only when the Match Memory is enabled (ENAB = 1).

The Global Data columns list both the output equation (input is determined by the slave controller) and the I/O switch controls. The F/S/B column designates Fast (F), Slow (S), or Broadcast (B) as the operating mode of the switch. The Input and Output functions require that the switch not be in the Broadcast mode. For those functions, fast or slow is determined by a previously executed Resolve function (such as Select First).

The Storage Part columns show the storage device input equations and the storage data latch (L) control signal (ENAB). A 1 indicates that the latch is enabled to capture a data bit from the storage device.

The Notes column provides explanatory comments for many of the function codes.

4.2.2.2 Match Memory -- The Match Memory (Figure 71) is a 16-word by 10-bit static RAM. Each of the 10 data bits is dedicated to a particular WLB. To minimize conductor paths on the WLLC, the inputs and outputs are combined into bidirectional lines by transceiver circuits. Word addressing is controlled by the four least-significant FC lines ($FC_{3,2,1,0}$). The control signals, Enable and Read/Write, are derived from the FC by the FC Decoding logic. Because the memory is static, no special clocking or refresh circuitry is required.

As mentioned earlier, the consolidation of match bit storage vectors into a random-access Match Memory is a means of achieving packaging efficiency. It does not limit the information processing or searching capabilities of the array.

4.2.2.3 Input/Output (I/O) Switch -- The I/O switch (Figure 72) is the data transfer element of the ECAM array. All data being transferred to or from the WLBs and data storage devices passes through an I/O switch. There is one switch on each WLLC.

4.2.2.3.1 Interfaces -- Each switch serves 10 WLBs. Each WLB supplies the switch with that block's match bit. (The same line also connects to the Match Memory.) The switch generates the pointer bit (P) which is part of the Multiple Match Resolver (MMR) logic. Ten bidirectional data lines connect the I/O switch to the WLBs.

All of the I/O switches are connected to a common I/O port. Conceptually, the 10 data I/O (DIO) lines from all switches are ORed together in parallel to form a 10-line port. In reality, the ORing is performed in stages at the hybrid, board, and cabinet levels by buffering logic.

The FC Decoding logic supplies each I/O switch with signals which control the operating mode of the switch.

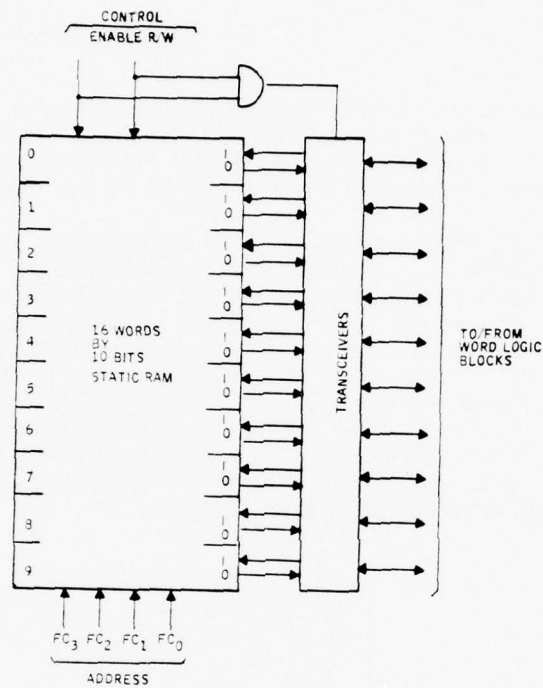


Figure 71. ECAM Word Logic Match Memory

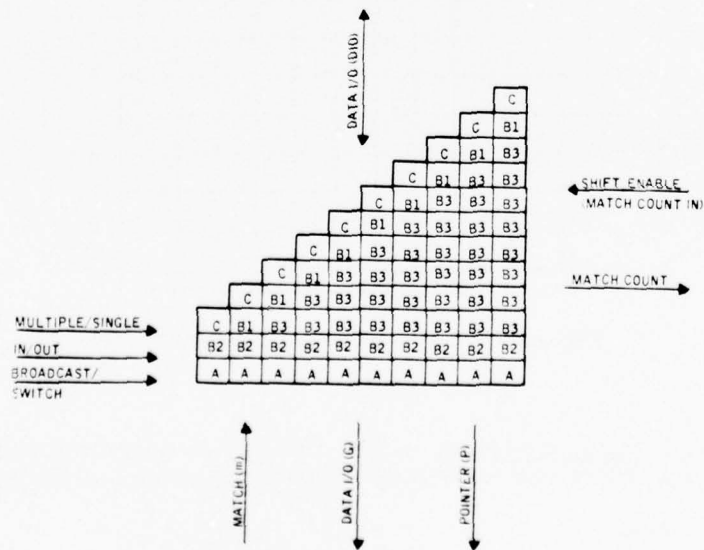


Figure 72. ECAM Word Logic I/O Switch

The I/O and MMR control logic supplies each I/O switch with shift enables which indicate the number of available DIO lines. In turn, the I/O switch supplies the I/O and MMR control logic with Match Count signals which indicate the number of data I/O lines required by the switch.

4.2.2.3.2 Operating Concept -- The operation of the I/O switch is shown conceptually in Figure 73. A switch matrix selects the "first" 10 participating words ($m = 1$) and switches them onto the first available I/O lines. For example, assume the words 4, 5, and 9 are participants. If an I/O switch higher in the array has already used one I/O line, then the switch pictured must route data from words 4, 5, and 9 to I/O lines 2, 3, and 4. The switch must also produce a Match Count value (equal to 3 in this case) which can be added to previous match counts, thereby providing a control input to subsequent I/O switches. The Match Count additions are performed by the I/O and MMR control logic and by the CGL tree. The P vector indicates the first (or first 10) responder(s) of the entire array.

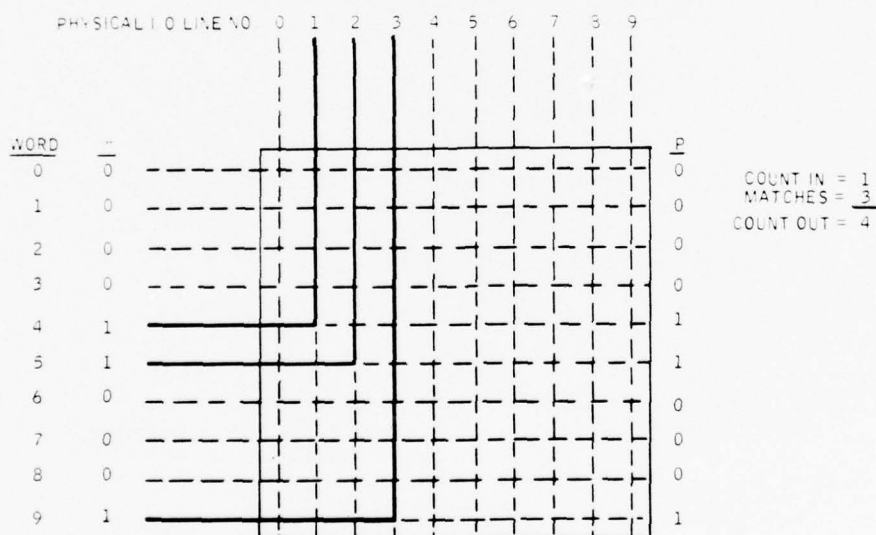


Figure 73. I/O Switch Operation Concept

4.2.2.3.3 Implementation -- The I/O switch operation concept is implemented as a triangular matrix which performs two functions:

- Selection
- Shifting

Participating array words are selected on the basis of the match bits (m). Selected Word Data I/O (WDIO) lines are connected to Logical Data I/O (LDIO) lines which are then shifted to Physical Data I/O (PDIO) lines.

The selector matrix is shown in Figure 74. The matrix is triangular because the first word, if matched, will always connect to the first logical data line. The second word can connect to either of the first two logical data lines, and so on. The selector matrix is controlled by a Connection Enable (CENAB) signal which propagates in a stairstep fashion through the matrix. Matched words cause the enable to be passed up and to the right on the next level. Non-matched words cause the enable to be passed to the right with no level change. The final level of the enable indicates the number of matches detected by the I/O switch.

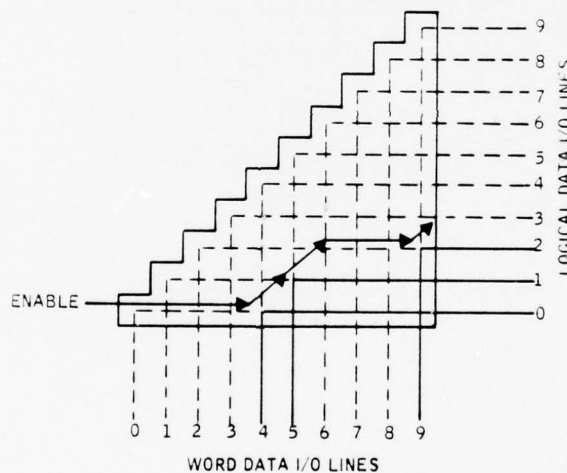


Figure 74. Selector Matrix

The shifter matrix is shown in Figure 75. The matrix is triangular because the first LDIO line may require connection to all 10 PDIO lines. The second LDIO line can be shifted to only nine of the PDIO lines, etc. The shifter matrix is controlled by 10 Shift Enable (SENAB) signals which run diagonally through the matrix. Only one Shift Enable is energized at a time. The value of the Match Count In controls selection of the proper Shift Enable. If more than 10 responders have been selected by the previous I/O switches, no Shift Enables will be energized.

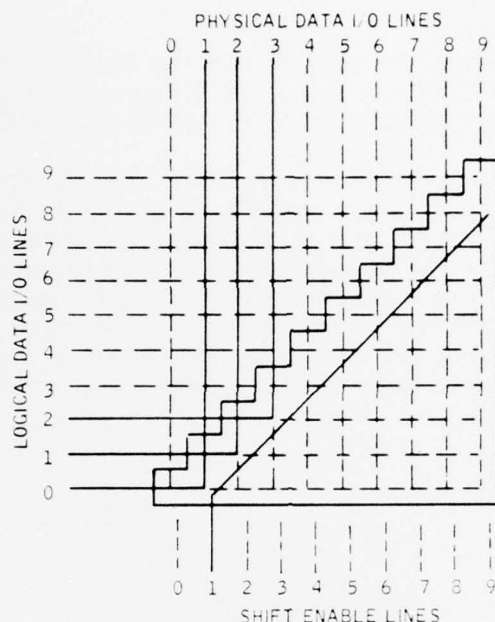


Figure 75. Shifter Matrix

The I/O switch is designed to perform four functions:

- Input
- Output
- Broadcast
- Output ORing

These functions correspond to the operating modes shown in Figure 76. Broadcasting is an Input mode and Output ORing is, of course, an Output mode.

The I/O switch allows both multiple (fast) and single (slow) input. In the Multiple Input mode, the "first" 10 participating ($m = 1$) array words are loaded simultaneously in bit-serial fashion from the 10 PDIO lines. If there are less than 10 participating words, then only the participating words will be loaded and unused PDIO lines will be ignored.

In the Single Input mode, the "first" participating ($m = 1$) array word is loaded in bit-serial fashion from the first PDIO line, PDIO₀. The remaining PDIO lines will be ignored.

CONTROLS			MODE
MULTIPLE/SINGLE	INPUT/OUTPUT	BROADCAST/SWITCH	
0	0	0	SINGLE OUTPUT
0	1	0	SINGLE INPUT
1	0	0	MULTIPLE INPUT
1	1	0	MULTIPLE OUTPUT
X	0	1	OUTPUT OR
X	1	1	BROADCAST

X = DON'T CARE

Figure 76. I/O Switch Mode Control

Selection of the "first" one (or 10) participating array words is performed by the I/O and MMR control logic operating in conjunction with the CGL tree.

The I/O switch allows both multiple (fast) and single (slow) output. In the Multiple Output mode, the "first" 10 participating ($m = 1$) array words are unloaded simultaneously in bit-serial fashion onto the 10 PDIO lines. If there are less than 10 participating words, then only the participating words will be unloaded.

In the Single Output mode, the "first" participating ($m = 1$) array word is unloaded in bit-serial fashion onto the first PDIO line, PDIO₀.

Selection of the "first" one (or 10) participating array words is performed by the I/O and MMR control logic operating in conjunction with the CGL tree.

In the Broadcast Input mode, all participating ($m = 1$) array words are loaded simultaneously in bit-serial fashion from the first PDIO line, PDIO₀. For proper operation, the remaining PDIO lines must be either zero or identical to the data on the first PDIO line.

Because the FC Decoding logic forces the first Shift Enable line (SENAB₀) to be true when the Broadcast mode is used, the I/O and MMR control logic and the CGL tree do not affect the operation of the switch during Broadcast Input mode.

In the Output ORing mode, all participating ($m = 1$) array words are ORed simultaneously in bit-serial fashion onto the first PDIO line, PDIO₀.

Because the FC Decoding logic forces the first Shift Enable (SENAB₀) line to be true when the Output ORing mode is used, the I/O and MMR control logic and the CGL do not affect the operation of the switch during ORING output.

The implementation of the I/O switch is realized by combining the selector and shifter matrices into a common I/O switch matrix with input and output controls. The matrix is constructed with three types of logic building blocks:

- Type A - Word Logic Interface
- Type B - Selector/Shifter
- Type C - Physical Data Interface

The blocks are interconnected as shown in Figure 72.

The Word Logic Interface block (Figure 77) converts the bidirectional DIO line (G) from a WLB into two unidirectional Word Data lines, WDI and WDO. It also provides buffering for the Pointer signal (P) and inversion of the match bit (m).

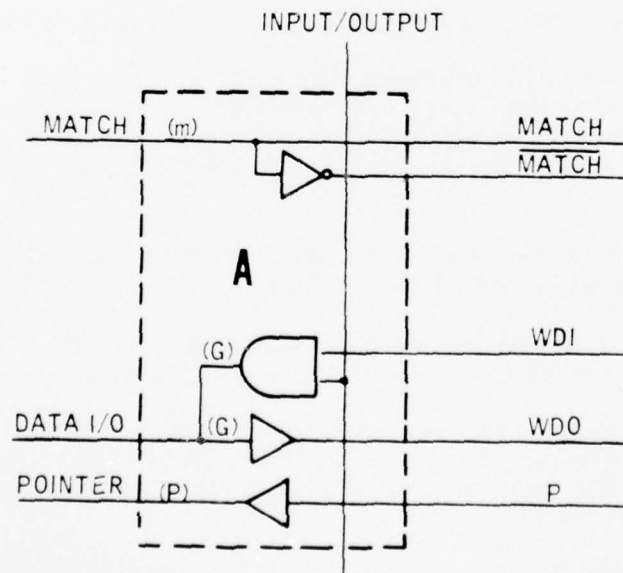


Figure 77. Word Logic Interface Block

The Selector/Shifter block exists in three nearly identical forms. Forms B1 and B3 (Figure 78) differ only in the source direction of the leftmost CENAB input. Form B2 (Figure 79) contains an additional gate (No. 10) which effects multiple/single control.

Selection and shifting, although co-located in the Type B blocks, are functionally distinct and operate independently. Selection is accomplished by Gates 1 through 5 and Gate 9. Gates 1 and 2 are used to control connection between a word and a LDIO line. If the word connected to the block has its match bit set ($m = 1$), and the LDIO line which passes through the block is the first one available, then the output of Gate 2 enables the connection and propagates the enable (CENAB) to the next LDIO (now the first one available). If the match bit is not set ($m = 0$), but the LDIO is available, then Gate 1 propagates the enable to the next block on the same LDIO. Gate 9 combines two CENAB sources for the next block. Gates 3 and 4 make the connection between the Word Data lines and the LDIO lines. Gate 5 generates the P pointer that enables the word itself (match resolution) if a PDIO is available.

Shifting is performed by Gates 7 and 8 which connect the LDIO lines to the PDIO lines when the SENAB line is true ($SENAB = 1$). Driver 6 generates a signal (LPC) indicating that a logical-to-physical connection has been made. This signal is necessary because not all of the words connected to LDIO lines may actually be connected to physical data lines by the shifter. The coincidence of match (m), CENAB, and LPC generates the Pointer signal, P.

In the Type B2 block, Gate 9 is used to introduce the Broadcast/Switch control line into the switch matrix. This line, when energized in conjunction with $SENAB_0$ causes the data received as input on the first (leftmost) PDIO line to be broadcast to all active ($m = 1$) WLBs. The data received on all the remaining physical data lines must be either zero or identical to the data on the first PDIO line. In the Output mode, energizing the Broadcast/Switch control line in conjunction with $SENAB_0$, causes the data supplied by all active ($m = 1$) WLBs to be ORed together and applied to the first (leftmost) PDIO, $PDIO_0$.

The PDIO line Interface block (Figure 80) converts the bidirectional PDIO line into two unidirectional Physical Data lines, PDI and PDO.

4.2.2.4 I/O and MMR Control Logic -- The I/O and Multiple Match Resolver control logic (Figure 81) generates control signals for the I/O switch. It also generates and combines Match Resolver signals in conjunction with the CGL tree.

The logical components of the I/O and MMR control logic are a decoder, an encoder, and an adder.

The decoder (Figure 82) is connected to the Match Count In (MCI) and Disable signal lines. It transforms those lines into the 10 SENAB lines which control the shifter matrix in the I/O switch. The decoder also provides an input for the Broadcast/Switch signal which is generated by the FC Decoding logic.

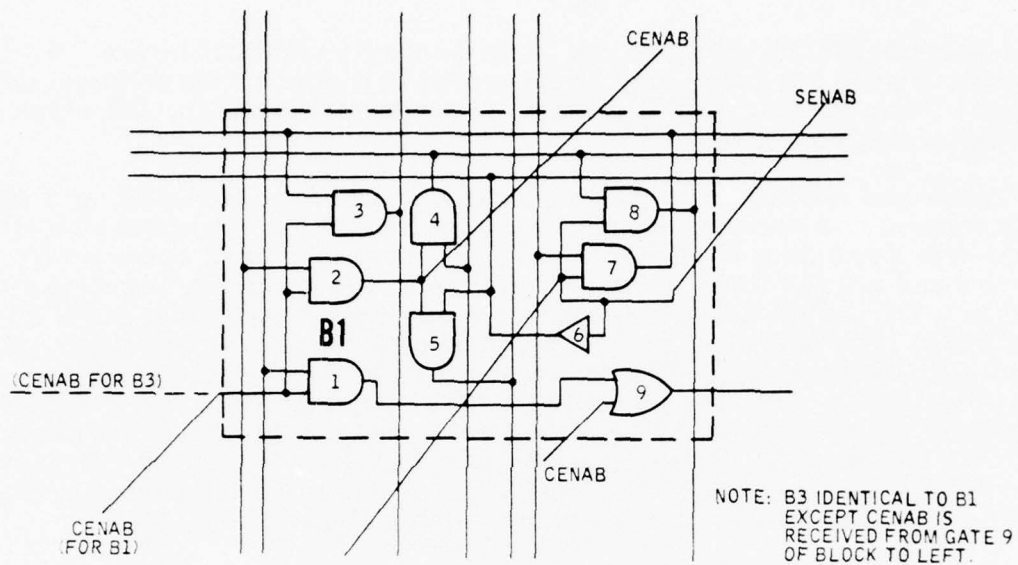


Figure 78. Selector/Shifter Blocks B1 and B3

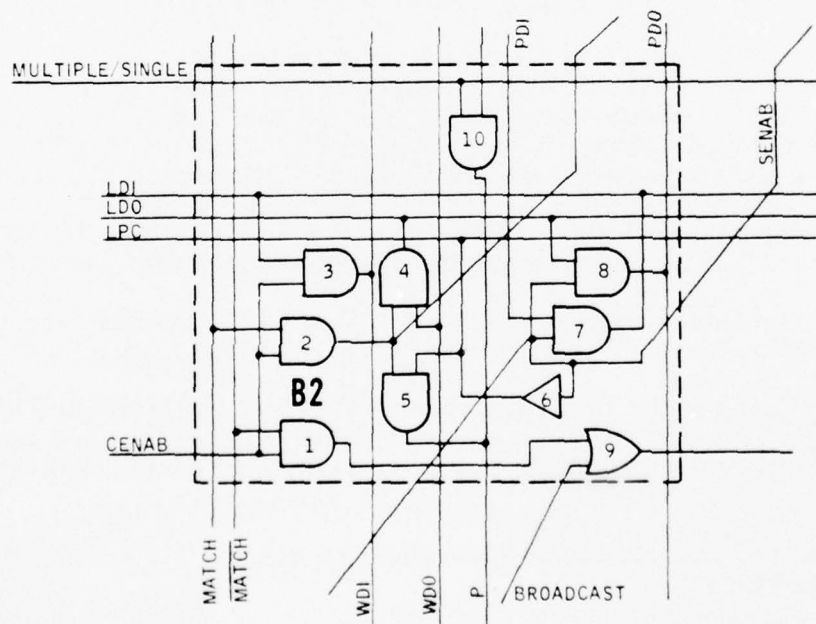


Figure 79. Selector/Shifter Block B2

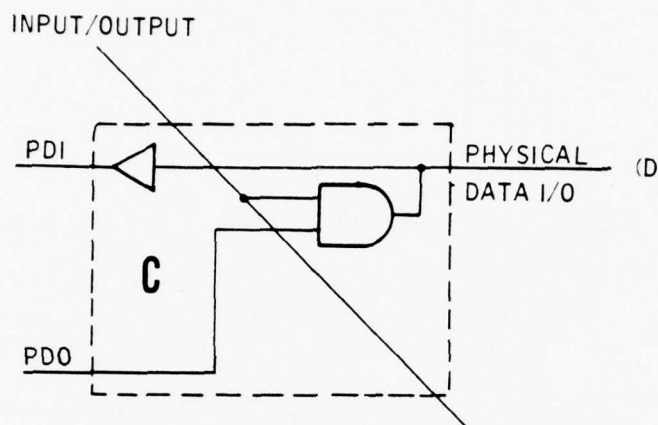


Figure 80. Physical Data Line Interface Block

The encoder (Figure 83) is connected to the final CENABs lines in the I/O switch. It transforms those 10 lines into four Match Count lines which are inputs to the adder. The encoder also contains the carry flip-flop and a control flip-flop, both of which are used during responder counting.

The adder (Figure 84) adds the value applied on the five Match Count In lines to the match count indicated by the encoder outputs. The sum is the Match Count Out. The adder consists of three full adder circuits, a half adder, and an OR gate.

The I/O and Multiple Match Resolver control logic controls the I/O switch during the following operations:

- Input/Output
- Multiple Match Resolution
- Responder Counting

During I/O and Multiple Match Resolution, the I/O and MMR control logic decoder transforms the Modulo 16 plus overflow representation of the number of responders "above" the controlled I/O switch into SENABs which result in the connection of Logical Data lines to unused Physical Data lines. Meanwhile, the encoder converts the CENAB signals, which indicate the number

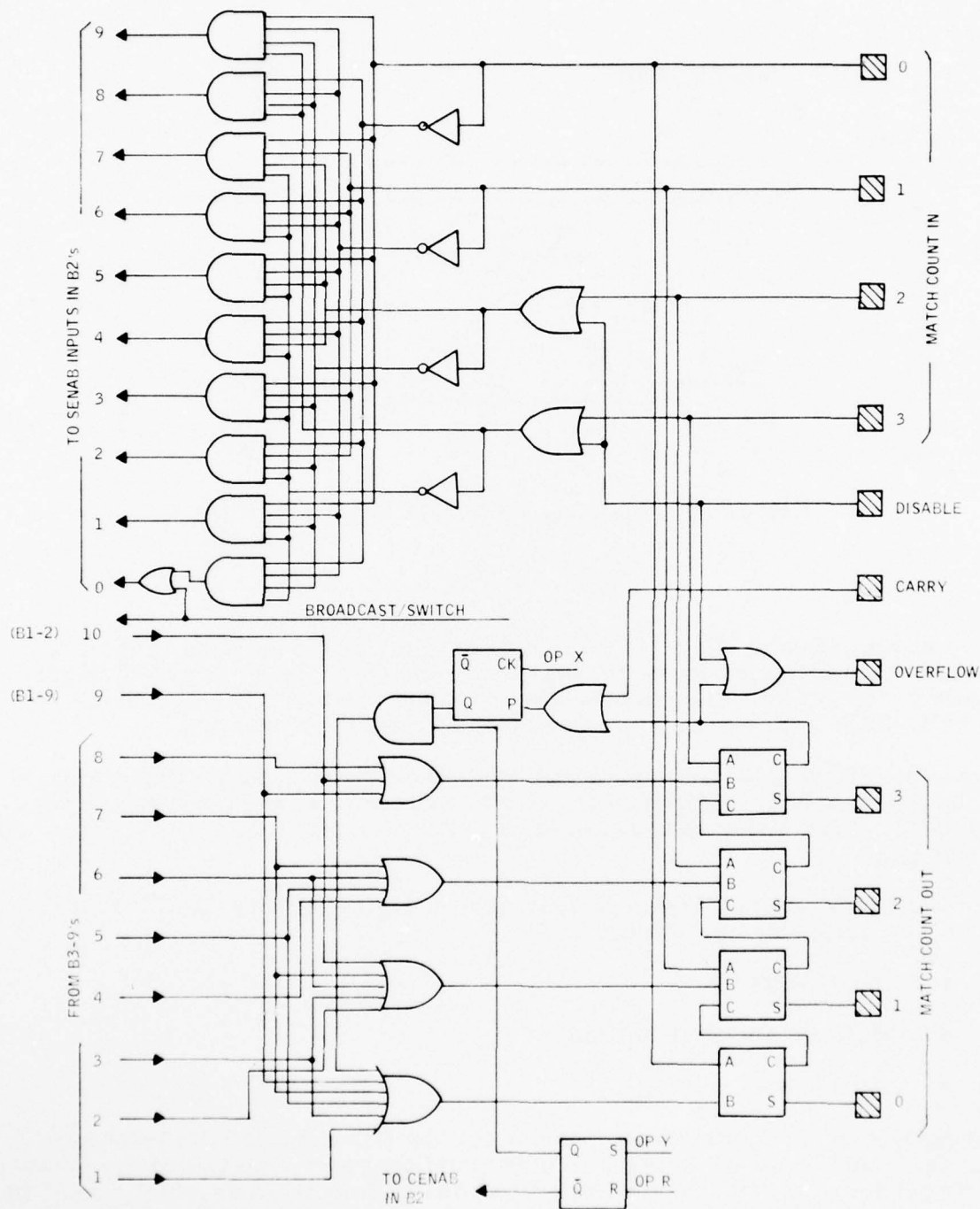


Figure 81. I/O and MMR Control Logic

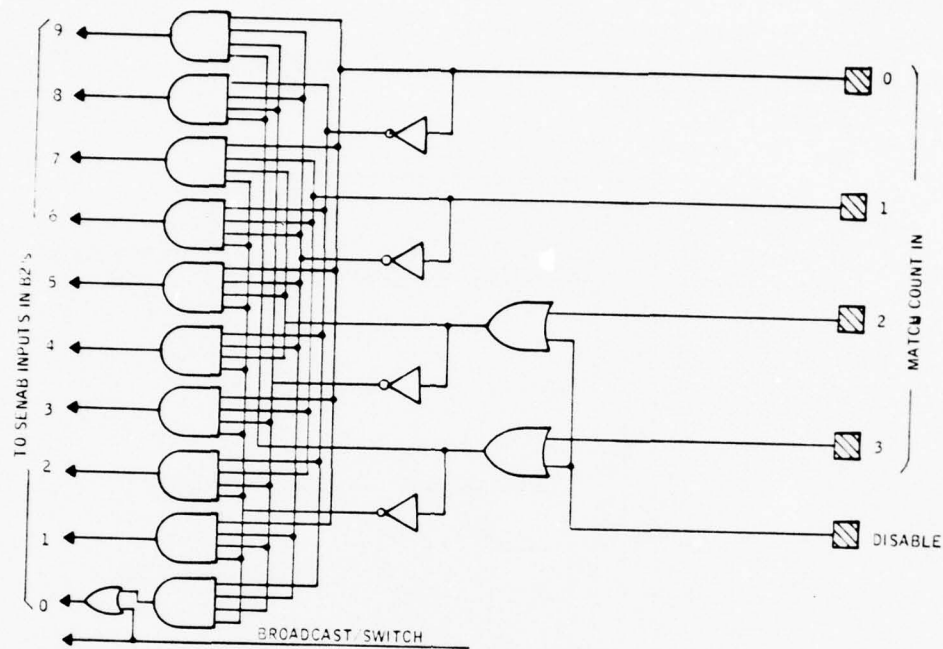


Figure 82. Decoder

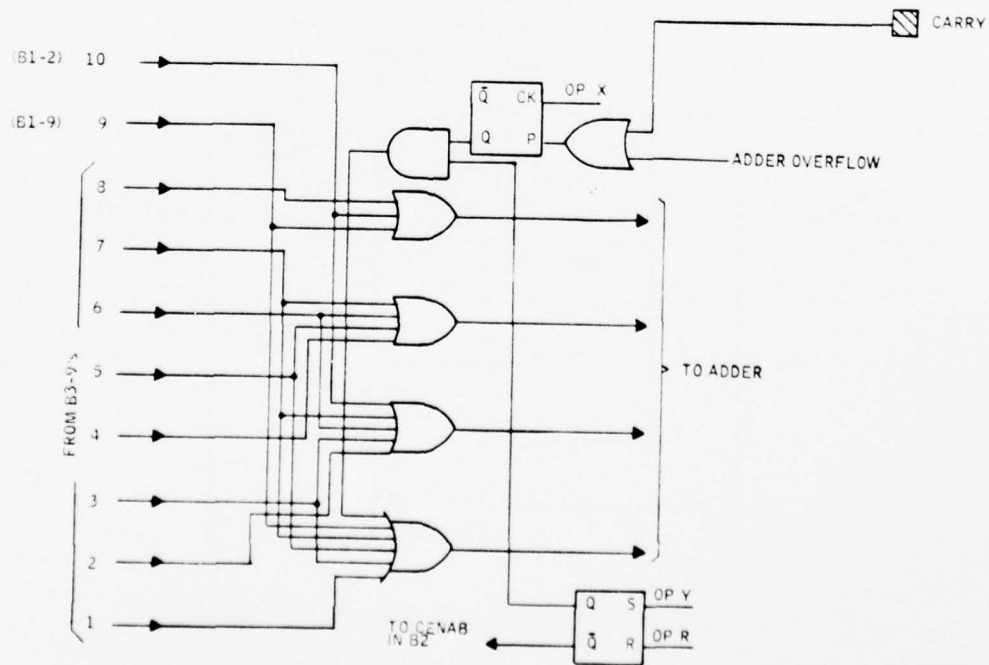


Figure 83. Encoder

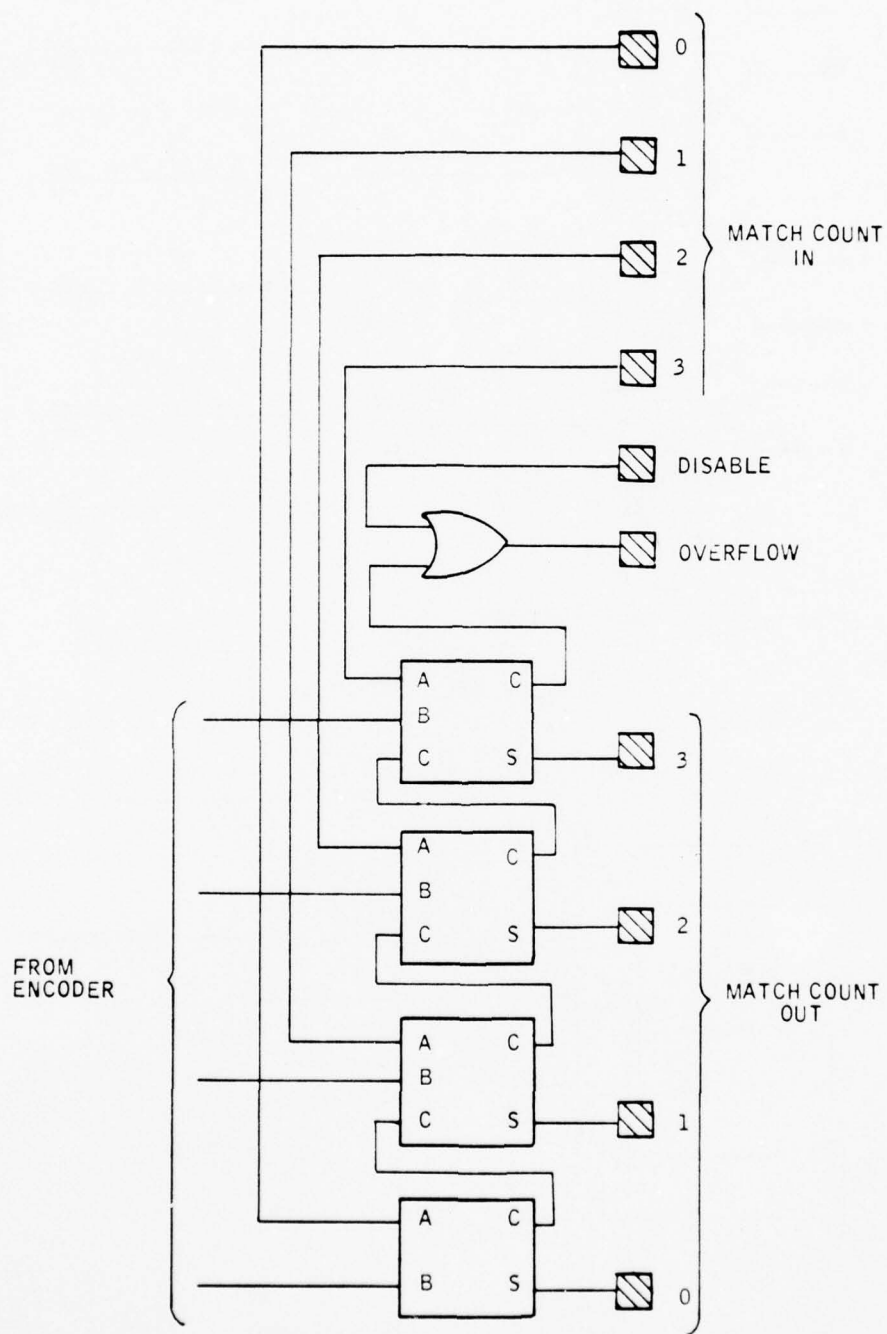


Figure 84. Word Logic Match Count Adder

of responders detected by the controlled I/O switch, to a binary representation which is used by the adder to compute the number of responders detected, including the particular WLLC.

During Responder Counting, the encoder first presents the number of detected responders to the adder for accumulation. After the slave controller has received the Modulo 16 result, the adder carries are captured by the capture flip-flop in each encoder. The control flip-flop is then set, causing the encoder to present the number of carries (0 or 1) to the adder for accumulation. Carries are captured and accumulated until no overflow exists at the slave controller. With each iteration, four bits of Responder Count value are recorded at the slave controller.

4.2.2.5 Function Code Decoding Logic -- The FC Decoding logic (Figure 85) is a combinatorial logic block which generates specific control signals required on the WLLC. Its function is analogous to that of the FC Decoding portion of the WLB.

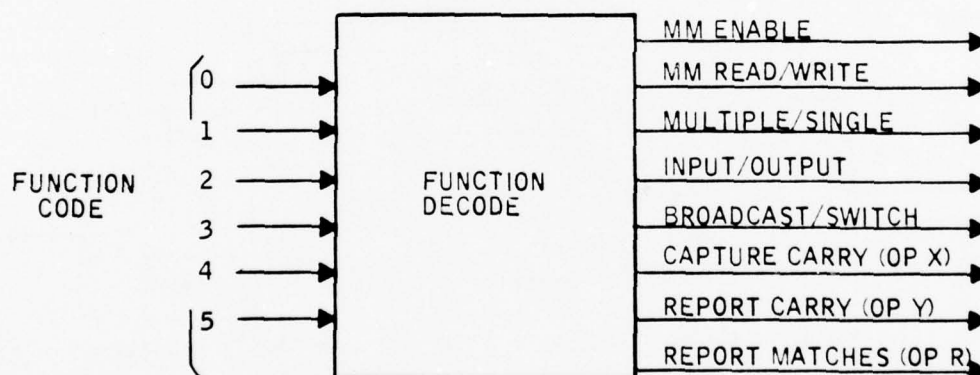


Figure 85. Word Logic Function Code Decoding Logic

4.2.3 Hybrid Circuit Module

The organization of a hybrid circuit module is presented in Figure 86. The components and their interconnection are discussed first. Then the operation of the hybrid during certain array operations is described.

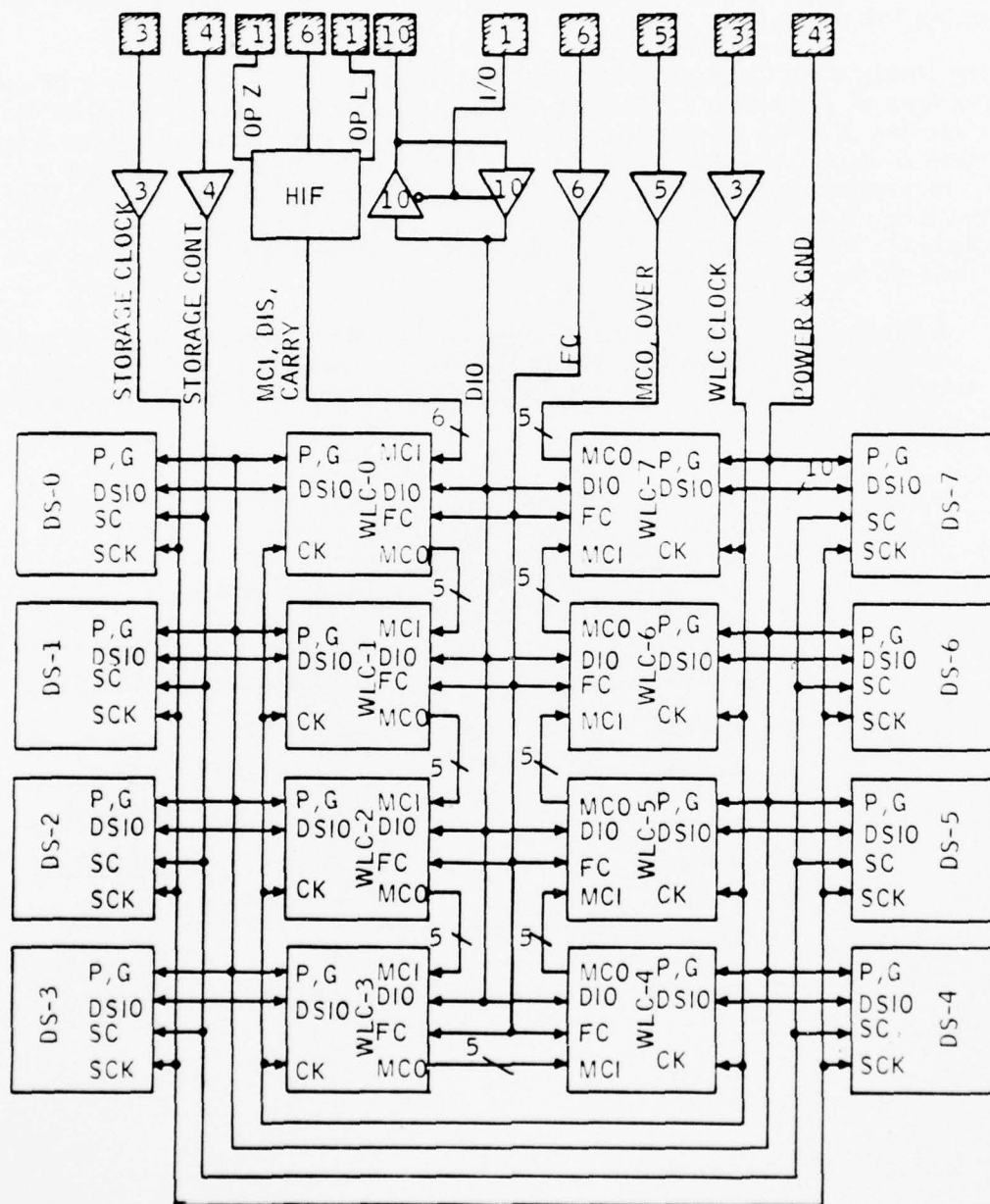


Figure 86. Hybrid Circuit Module

Each hybrid circuit module contains eight sets of WLLC Data Storage chip combinations. Thus, a single module provides 80 ECAM array words. The module also contains signal buffering circuits which are standard, commercially available chips.

In addition to the usual power and ground distribution, the hybrid circuit module contains interconnections for clock, data, and control.

Two clock distributions are provided. The Storage Clock lines serve the Data Storage chips. A separate set of Clock lines is used by the WLLCs (WLC Clocks). Baseline storage devices are shown in the figure. Alternative devices may have different clocking requirements. Ten bidirectional DIO lines (called PDIO in the I/O switch) are wire-ORed from each WLLC. Bidirectional buffers (transceivers) are used at the hybrid connector pins. Control of the Data Storage chips is supplied by four Storage control lines which are connected directly to those chips. Function control of the WLLCs is supplied by the six FC lines.

I/O switch and MMR control is both supplied to and generated in the hybrid circuit module. The Hybrid Interface (HIF) circuits capture the Match Count In and Disable control signals to the hybrid (Carry is captured in the first WLLC). Additionally, each WLLC generates four Match Count Out signals and a Disable signal which are connected to the next chip. The outputs from the last chip are supplied to the CGL tree.

The hybrid circuit module uses function control signals and I/O/MMR control signals. Function Codes (FC) applied to each WLLC control both the actions performed by the WLBs, and also the operating mode of the I/O switch. These functions are described in Section 4.2.2.1. The I/O/MMR control signals are supplied by the CGL tree (board level). They consist of Match Count In, Disable, and Carry In. The controls support the operations of Input/Output, MMR, and Responder Counting.

The Match Count In is four lines which are binary coded to represent the number of responders which have been detected "above" the WLLC to which the count is applied. The I/O switch will not connect a WLB to an I/O line when the Match Count In is greater than 10. Each WLLC adds the number of responders ($m = 1$) it detects to the Match Count In to produce the Match Count Out. The Match Count Out becomes the Match Count In of the next WLLC. The final Match Count Out informs the CGL tree of the number of matches detected by the hybrid. NOTE: Match Count In to a hybrid and Match Count Out from a hybrid are not used simultaneously. Section 4.2.4 describes the operation at a higher level. The Match Count In signal to a hybrid is not used during the Responder Count operation.

The Disable control signal is generated (by the CGL tree or by a WLLC when the accumulated Match Count exceeds 15. Once generated, the Disable control signal is propagated to the end of the array. This signal is necessary because the Match Count lines contain the result of Modulo 16 additions. The Disable signal is not used during the Responder Count operation.

During the Responder Count operation, the Carry In signal to the hybrid indicates that a carry has occurred in the CGL tree. The carry will be captured in the first WLLC and counted on the next iteration. The Carry In pin of each of the remaining seven WLLCs in the hybrid is forced to a logic zero. Adder carries in those chips will be captured within each chip. The Carry In signal is ignored during Input/Output and MMR operations.

4.2.4 Array Storage Board

The organization of an array Storage Board is presented in Figure 87. The components and their interconnection are discussed below.

Each Storage Board contains:

- Eight hybrid circuit modules
- A CGL tree (Levels 1, 2, and 3)
- Function Code decoding and buffering
- Interconnect signal buffering

One board comprises the storage, processing, and control hardware for 640 array words (each 4096 bits in length).

One hybrid circuit module contains 80 Data Storage words and 80 WLBs (processing elements). The construction and operation of the hybrids are discussed in Section 4.2.3.

The CGL tree (Figure 88) controls the operation of the I/O switches and provides the MMR capability for eight hybrid circuit modules. It also generates Responder Address signals which identify the first hybrid circuit module which contains a participating ($m = 1$) array word.

The FC Decode and Buffering logic is a read-only memory (ROM) or programmable logic array (PLA) which generates board and hybrid-level control signals based on the FC lines. This logic also serves as a buffer for those lines and thereby aids signal distribution.

The Interconnect signal buffering reduces the signal load requirement created by the card. It also increases the drive and ORing capability of signals leaving the card.

In addition to the usual power and ground distribution, the array circuit board contains interconnections for clock, data, and control.

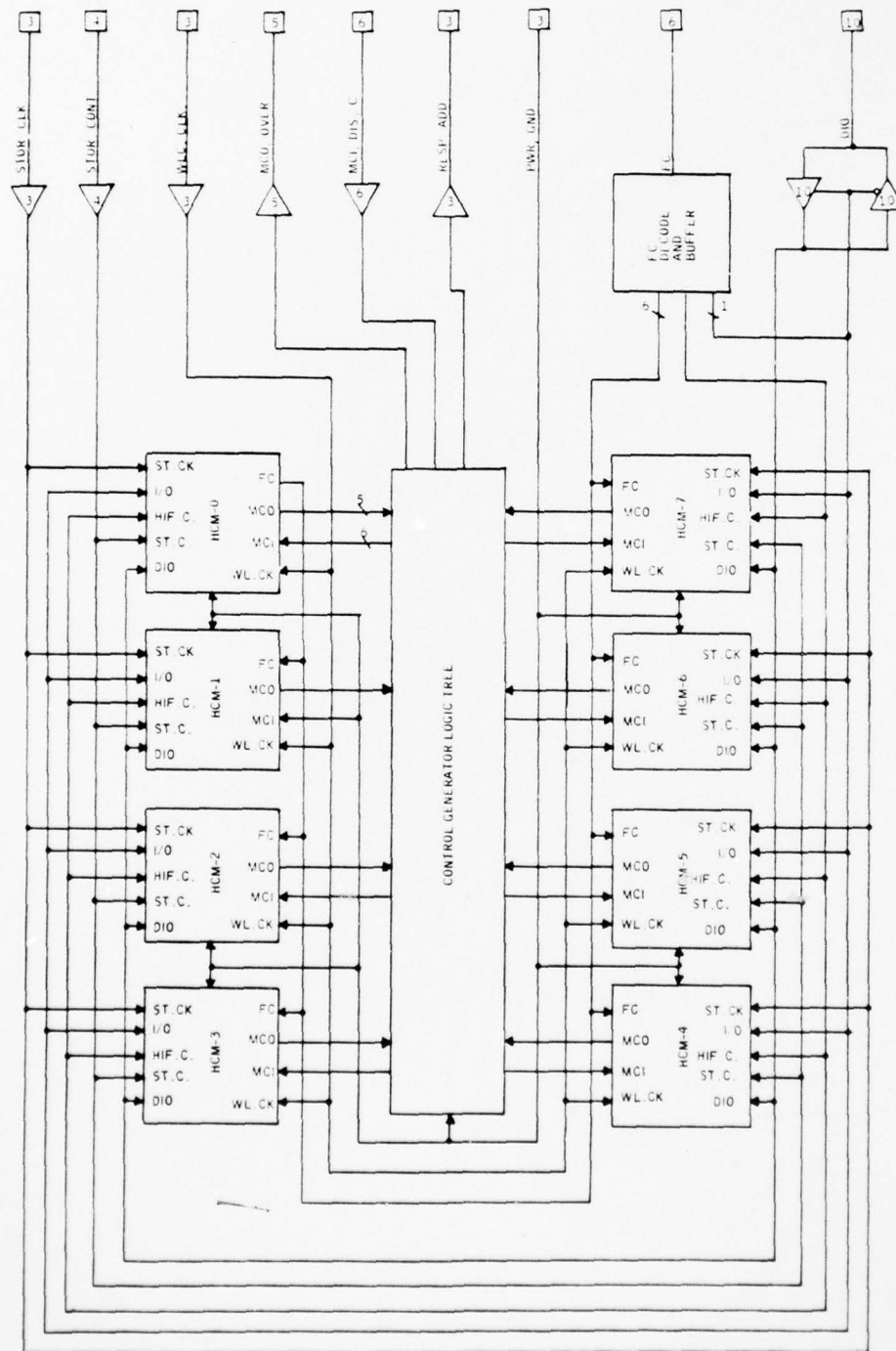


Figure 87. Array Storage Board Function Block Diagram

AD-A037 834

HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
ADVANCED LOGIC TECHNOLOGY.(U)

FEB 77 G A ANDERSON, E D JENSEN, R Y KAIN

F30602-75-C-0148

UNCLASSIFIED

F0375-FR

RADC-TR-76-388

NL

3 of 4
ADA037834



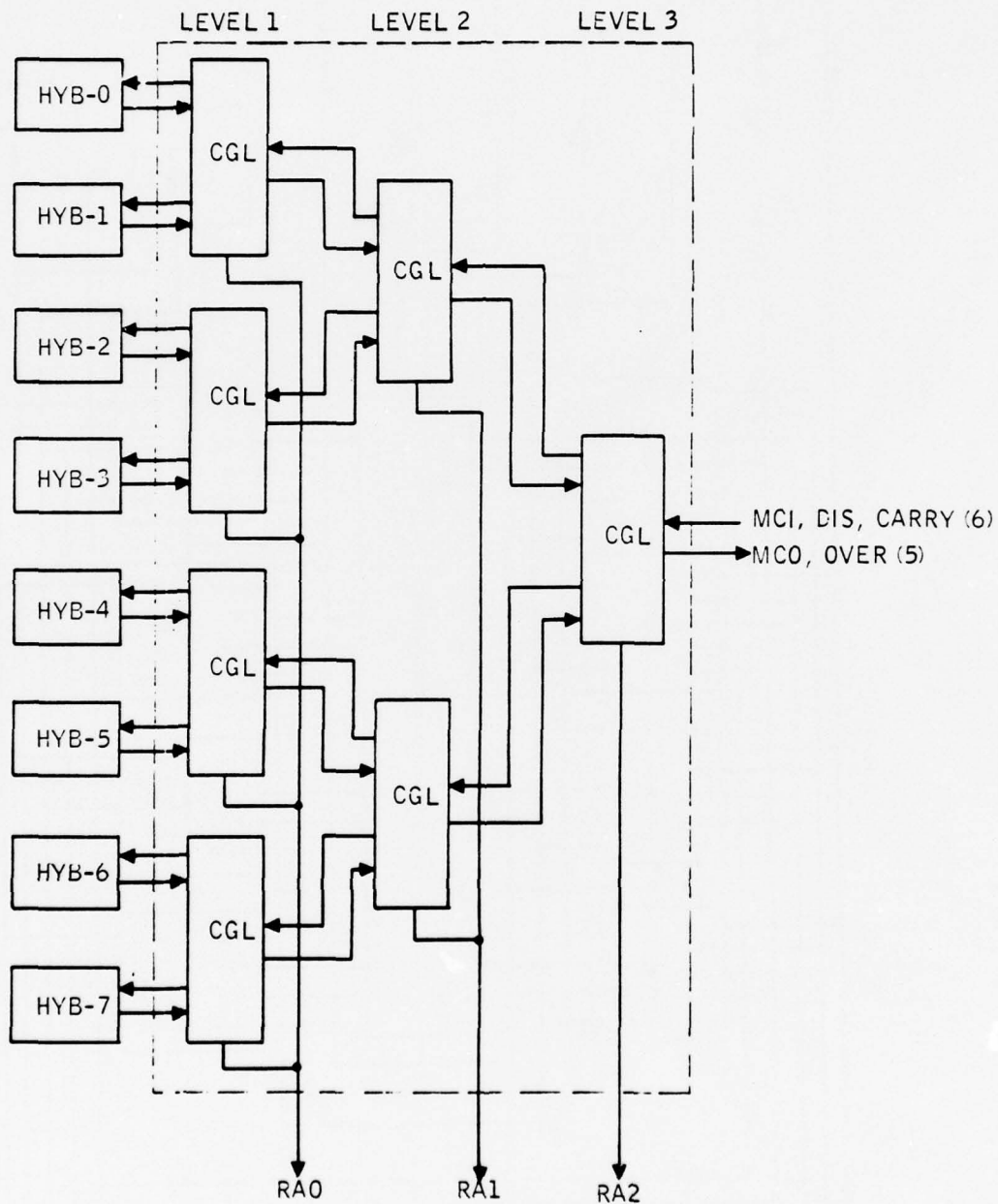


Figure 88. Storage Board Control Generator Logic Tree

Two identical clock distributions are provided. Each consists of a set of three lines and each is connected to the hybrid circuit modules. One set serves the Data Storage chips, while the second serves the WLLCs.

Ten bidirectional DIO lines are wire-ORed from each hybrid circuit module. Bidirectional buffers (transceivers) are used at the card connector pins.

Control of the Data Storage chips within the hybrid circuit modules is provided by four Storage Control lines which are connected to each hybrid. Function control of the WLLCs is supplied by six FC lines which are also connected to each hybrid. The FC Decoding logic generates an I/O control line which is used by the hybrids and the data transceivers. In addition, the FC Decoding logic produces two controls for the HIF logic within the CGL tree, and buffers the six FCs before they are applied to the hybrids. I/O switch and MMR control signals are created within the CGL tree and applied directly to each hybrid. Eleven control signals (six in and five out) are used to connect the circuit board to the more global levels of the tree (cabinet buffers, etc.). The Responder Address lines (3) which originate in the CGL tree are sent to the card connector for combination with signals from other cards and used by the slave controller.

4.2.5 Control Generator Logic (CGL) Tree

This logic structure deserves additional discussion. In the following paragraphs, the structure of the tree is presented first, then the overall operation is described, and finally, the operation of the individual components is explained.

4.2.5.1 Structure -- The structure of the CGL tree is evident in Figure 88. Packaging constraints require that the number of chips contained in each hybrid circuit module be minimized. Therefore, the tree structure is not continued within the hybrids. Instead, a ripple technique is used to control the I/O switches within the hybrid. Refer to Section 4.2.3 for details. The result of this break in the uniformity of the tree structure is that HIF circuits are required. These circuits (located within the hybrid) consolidate the operational differences that exist between the tree structure and the ripple structure.

Identical building blocks are used to implement the branch nodes of the tree. The tree structure is continued by interconnecting array Storage boards to similar logic on the Storage Buffer and Cabinet Buffer boards and at the subsystem level. The logic at the higher levels is not described in this section. It is merely a continuation of the tree structure.

The base of the tree is connected to the slave controller. It indicates the Modulo 16 count of the number of active (responding) array words. The tree

also supplies the slave controller with the address of the first responding hybrid. The Address lines (RA_0, RA_1, \dots, RA_n) are wire-ORed across array circuit boards.

4.2.5.2 Overall Operation -- The CGL tree provides control environment for the hybrid circuit modules. The operations controlled by the tree include:

- I/O Preparation and Multiple Match Resolution
- Responder Counting
- Responding Hybrid Address Generation

4.2.5.2.1 I/O Preparation and Multiple Match Resolution -- The result of this operation is the selection of one array word or a set (maximum 10) of array words. The word (or words) selected is the first of the matched ($m = 1$) words of the array. While the actual selection is performed by the I/O switches, the selection is based on the outputs of the CGL tree.

The tree receives a match count from each hybrid indicating the number of matched array words within the hybrid. The tree combines the match counts to produce a series of partial sums, one for each hybrid. A partial sum (called *Match Count In*) indicates to a hybrid the number of matched words which have been detected in previous hybrids. The partial sums are represented Modulo 16 with an overflow indication.

During I/O Preparation, either one or a set (maximum 10) of words is selected. This is controlled by the Multiple/Single control derived from the FC. During MMR, usually one word (the first) is selected, again using the Multiple/Single control. The two functions are the same. Only the objective may be considered different.

The tree structure is fully combinatorial and does not, if fully implemented, require a sequence of operations to reach its final state. However, because the tree is not implemented within the hybrid circuit modules (a packaging restriction), and because the hybrids do not follow the interconnection rules of the tree, a two-step operating sequence is required.

First, the HIF registers are cleared. This causes the Match Count In of each hybrid to equal zero. The hybrids will each simultaneously accumulate their own Match Count sum by ripple adding the count of each WLLC. When the sums reach the Match Count Out lines of each hybrid, the tree begins generating partial sums by using a hierarchical adding technique. When the partial sums are present at the inputs of the HIF registers, the first operating step is complete.

As the second step, the HIF registers are loaded with the partial sums generated by the tree. This causes each Match Count In to equal the sum of the previously supplied Match Count Out values, or

$$MCI_n = \sum_{i=0}^{n-1} MCO_i$$

The hybrids will then simultaneously accumulate new partial sums internally until the Match Count In of each WLLC is correct. The final Match Count Out of the hybrid will be supplied to the CGL tree, but is not used.

4.2.5.2.2 Responder Counting -- The object of the Responder Counting operation is to inform the slave controller of the total number of matched ($m = 1$) words within the array. The counting is performed by an iteration process that produces four bits of result at the slave controller with each iteration.

To initialize the array, the HIF registers are first cleared. This causes the Match Count Out generated by each hybrid to indicate the Modulo 16 representation of the number of matched words within the hybrid. The tree combines the Match Count in successive stages until a four-bit result is produced at the base of the tree. The four bits represent the least significant four bits of the responder count. If the Overflow signal at the base of the tree is zero, the counting is complete.

In the summing process, carries are generated at each point in the tree where the sum exceeds 15. These carries are propagated back toward the hybrids. The Capture Carry FC causes the carries to be latched in flip-flops in the hybrids. Tree-generated carries are captured in the first WLLC of a hybrid. Carrys which are generated within the ripple adder structure of a hybrid are simultaneously captured in the seven remaining WLLCs in each hybrid.

After the carries have been captured the Report Carrys function is executed. This FC disables the I/O switch so that it does not detect matches. The Carry Capture flip-flop then presents its value (1 or 0) to the adder. A Modulo 16 sum of carries is then generated by the adders and may be removed by the slave controller at the base of the tree. This sum represents the next four bits (more significant than the previous four) of the Responder Count. If the Overflow signal at the base of the tree is zero, the counting is complete. If not, carries are again captured and another four bits of sum are generated.

When the counting is complete, the Report Matches function is executed. This restores the I/O switch to its normal match-detecting state. Because four bits of sum are generated with each iteration, no more than $I = (\log_2 N)/4$ iterations are required, where N is the number of responders (matches) and I is rounded upward to an integer value.

During I/O Preparation and Multiple Match Resolution, the CGL tree indicates, in binary, the physical address of the hybrid circuit module which contains the first responding ($m = 1$) array word. The address is present on the Responder Address lines (RA_0, RA_1, \dots, RA_n) after the HIF registers have been cleared but before they are reloaded.

4.2.5.3 Component Operation -- The HIF consolidates the operating differences between the CGL tree and the hybrid circuit modules. One HIF block is shown in Figure 89. This logic is located in the hybrid circuit modules.

When the Zero Hybrid Interface Register FC (OP Z) is executed, all HIF flip-flops are cleared. When the Load Hybrid Interface Register FC (OP L) is executed, all HIF flip-flops are simultaneously loaded with the code value supplied by the CGL tree.

The Control Generator Logic block (Figure 90) is the basic unit of the CGL tree. Each block connects to one block on the right (trunk or base side) and to two blocks on the left (leaf side).

The block contains two adders and two OR gates which are used for most functions. The adder and gate at the lower left combine the Match Count and Overflow signals from two sources (blocks or hybrids) and pass the result to the right. An Overflow signal from this adder constitutes a carry which is passed to the hybrids for capture during Responder Counting. The adder and gate at the upper right combine the Match Count and Overflow signals from the upper source with the Code and Disable signals from the block on the trunk side to produce Code and Disable signals for the block or hybrid to the lower left (leaf side). The Code and Disable signals contain partial sum information like the Count and Overflow signals, but they propagate toward the leaves rather than toward the trunk of the tree. At the base (trunk) of the tree, the Carry, Code, and Disable signals are hardwired to zero.

The three remaining gates detect the occurrence of no responses above with one or more responses below. This serves as a binary Responder Address line. All Responder Address lines generated at the same tree level are ORed together.

4.2.6 Associative Array Summary

The elements of the ECAM Associative Array have been presented and discussed. Word Logic Blocks provide each Storage Word with a processing capability that supports Associative array functions. A RAM supplies the WLBs with a workspace for intermediate-result storage during evaluation of complex search expressions. A unique switch mechanism enables high-speed input and output. The I/O switch combines with a tree-structured control scheme to perform Multiple Match Resolution and Responder Counting. A hierarchy of signal distribution and buffering is mated with the CGL tree to take advantage of the various packaging techniques and thereby assure an easily implemented and realizable array structure.

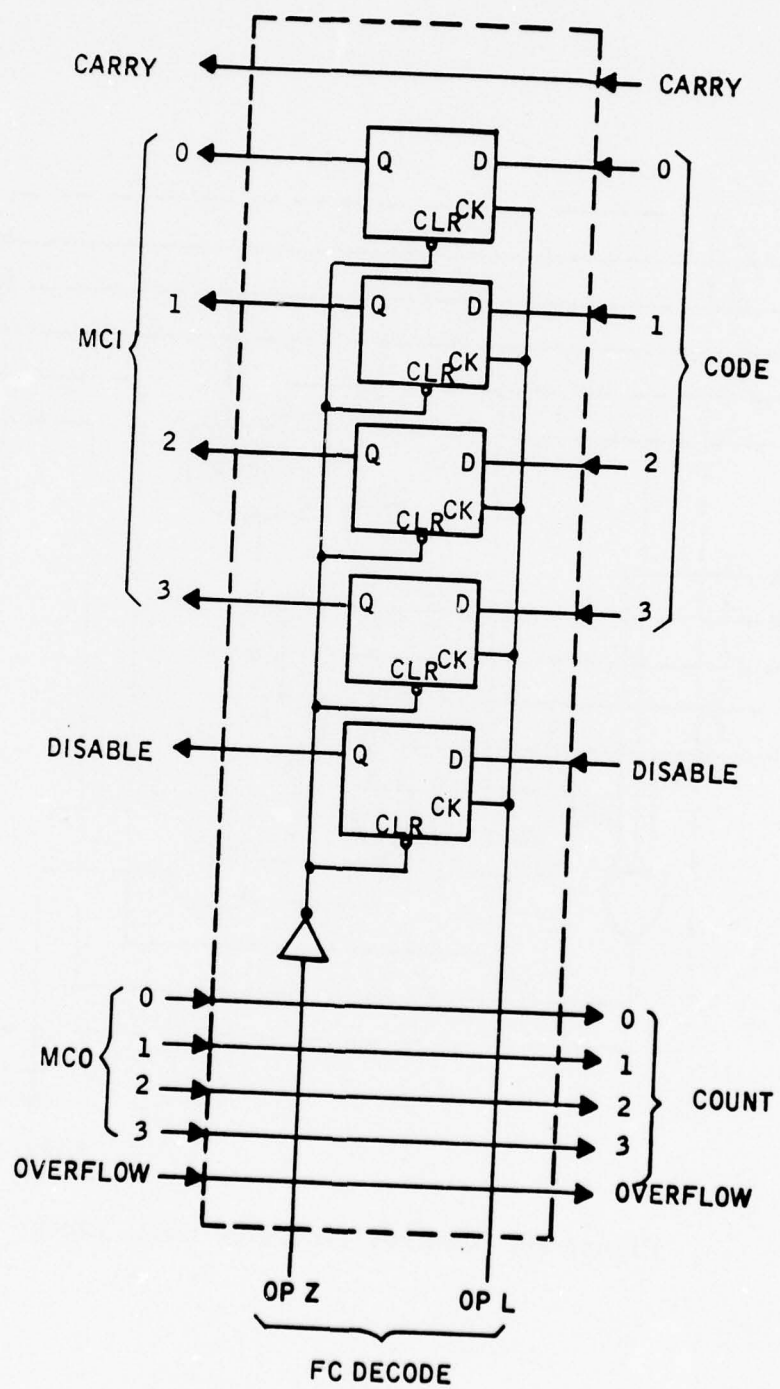


Figure 89. Hybrid Interface Block

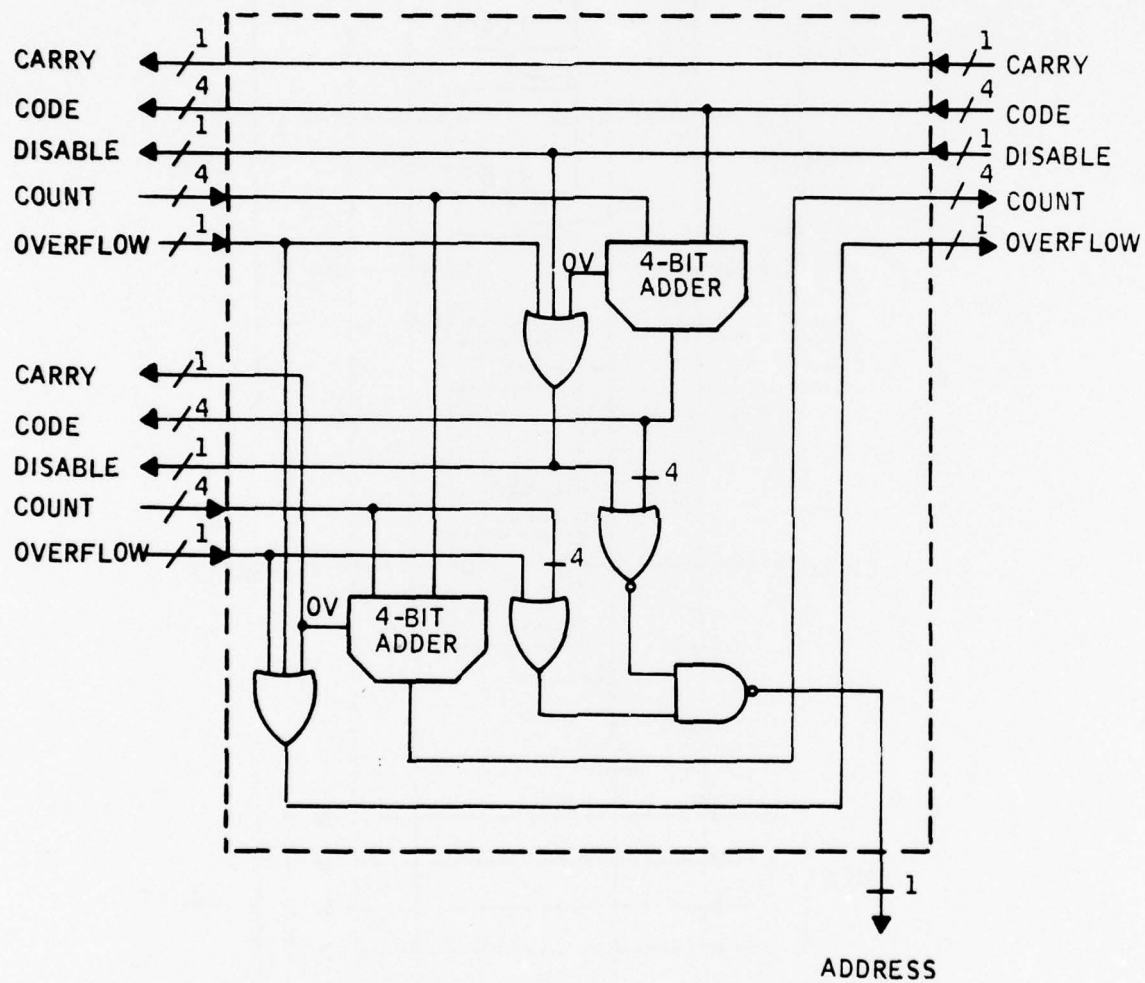


Figure 90. Control Generator Logic Block

SECTION 5

SYSTEM OPERATION

This section describes a candidate mapping of the SACWARDANS directories onto the ECAM hardware and discusses the slave microprograms which provide access to the data base. The SACWARDANS mapping is provided as a context for discussion; the final mapping will depend on the data requirements when the ECAM is installed and will be the result of a series of storage efficiency/speed tradeoffs. The microprogram approach, however, provides a general-purpose relational environment which will support a wide range of applications. Changes to the approach are expected to result from actual user experience, rather than from additional a priori design work.

5.1 SACWARDANS DIRECTORY STORAGE

This section describes the mapping of the SACWARDANS data base (as described in Section 2) onto the ECAM hardware. It was prepared by Informatics, Inc. as part of the analysis work done early in the contract. As such, it is in slight variance with the final design. Specifically, it describes a contiguous format field containing scratch storage when, in fact, this information is distributed across words as described in Section 5.4. These format changes have virtually no impact on the discussions which follow.

5.1.1 SACWARDANS Directories Organization in ECAM Array

Search time and algorithm complexity are reduced, with the current ECAM word architecture, by formatting the ECAM words as a series of different record types. Figure 91 illustrates this approach. The problem that arises

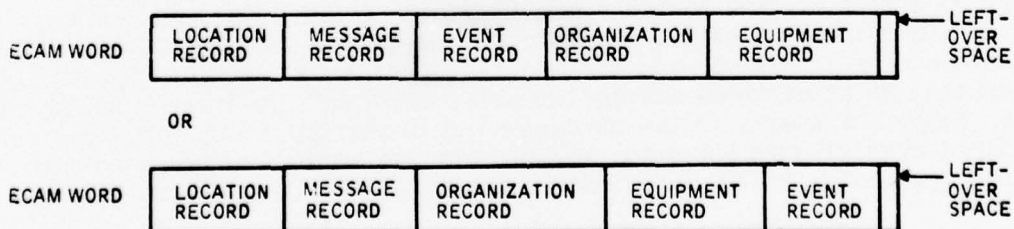


Figure 91. ECAM Word Formatting Approach

in attempting to format with a single instance of each record type in an ECAM word is that it may not be possible to fill the words, or even come close to filling the words. As shown in Figure 91, one instance of each record type less than half fills an ECAM word. In addition, the wide disparity in the expected number of entries in the various directories would result in further unused space. This occurs as the result of not using space allocated to a particular record type because there are insufficient instances of the record type as compared with the number of instances of another record type.

5.1.1.1 SACWARDANS Formats -- The SACWARDANS directories data base contains a large number of message and event correlation records and relatively few other records. Consequently, it is necessary to pack several of each of these records into a single ECAM word. Otherwise, the amount of wasted space would be very high (greater than 80 percent of the memory when a word length of 512 bytes is used). Figures 92 through 96 show five formats for ECAM word layouts for use with the SACWARDANS directories data base. The first format, shown in Figure 92, contains only message and event records. The other formats use the same basic structure but the space occupied by one message and one event record is allocated to other record types. All five formats would be required in order to accommodate the SACWARDANS directories data base. Formats 2, 3 and 5 would be used to record the Equipment, Organization, and Location Indexes. The L/O/E Index appears in formats 2, 3, and 4. These formats would be used to the extent required to accommodate the instances of these lists. Format 1, which contains only message and event records, would be used to contain the remaining message and event instances. The mix of formats required for the SACWARDANS directories data base as previously described is:

- Format type 1 occurs in 60,000 instances.
- Format type 2 occurs in 1000 instances.
- Format type 3 occurs in 5000 instances.
- Format type 4 occurs in 9000 instances.
- Format type 5 occurs in 5000 instances.

Figure 97 shows all five formats in a single diagram. The five formats are structurally similar. All have a format field in the same position. All have six message-type records and six event-type records. Again, the structuring is such that these records occupy the same columnar position in all five formats. Hence, a search of the message list is performed by the same search algorithm over all five formats, except that one additional record position must be applied to words in format 1 (which has a seventh message record).

5.1.1.2 Format Field -- Figures 92 through 96 show a format field in all of the layouts. This field serves three purposes: identification of which format a particular word is in; markers indicating which records within the format are actually in use; and provision of work space for use by algorithms. Thirty-six bytes are shown for this purpose. This may be more than necessary, but it should offer a flexible capability.

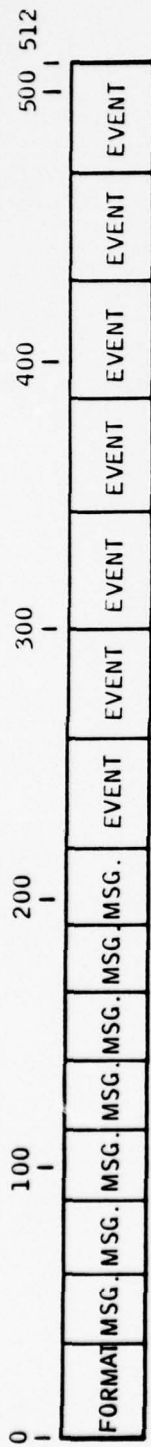


Figure 92. Format 1

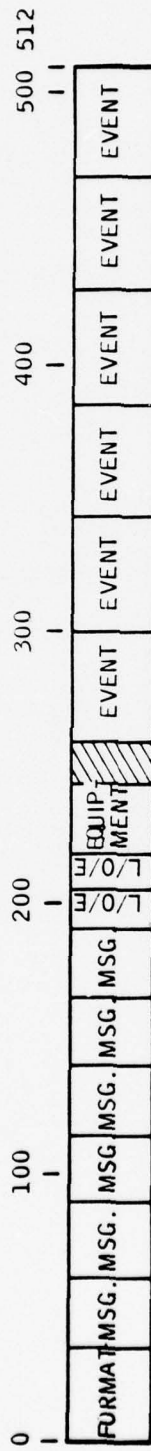


Figure 93. Format 2

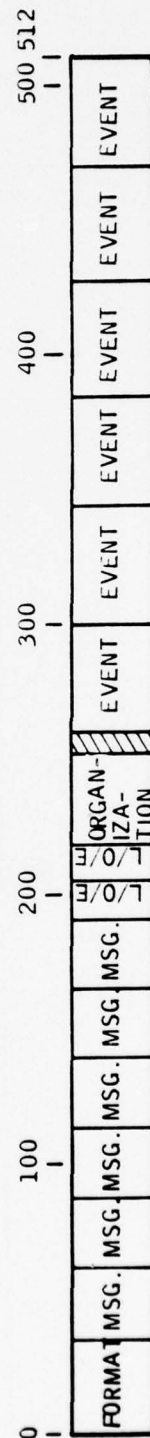


Figure 94. Format 3

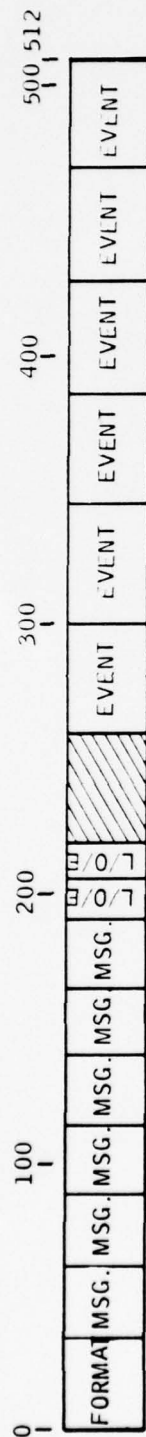


Figure 95. Format 4



Figure 96. Format 5

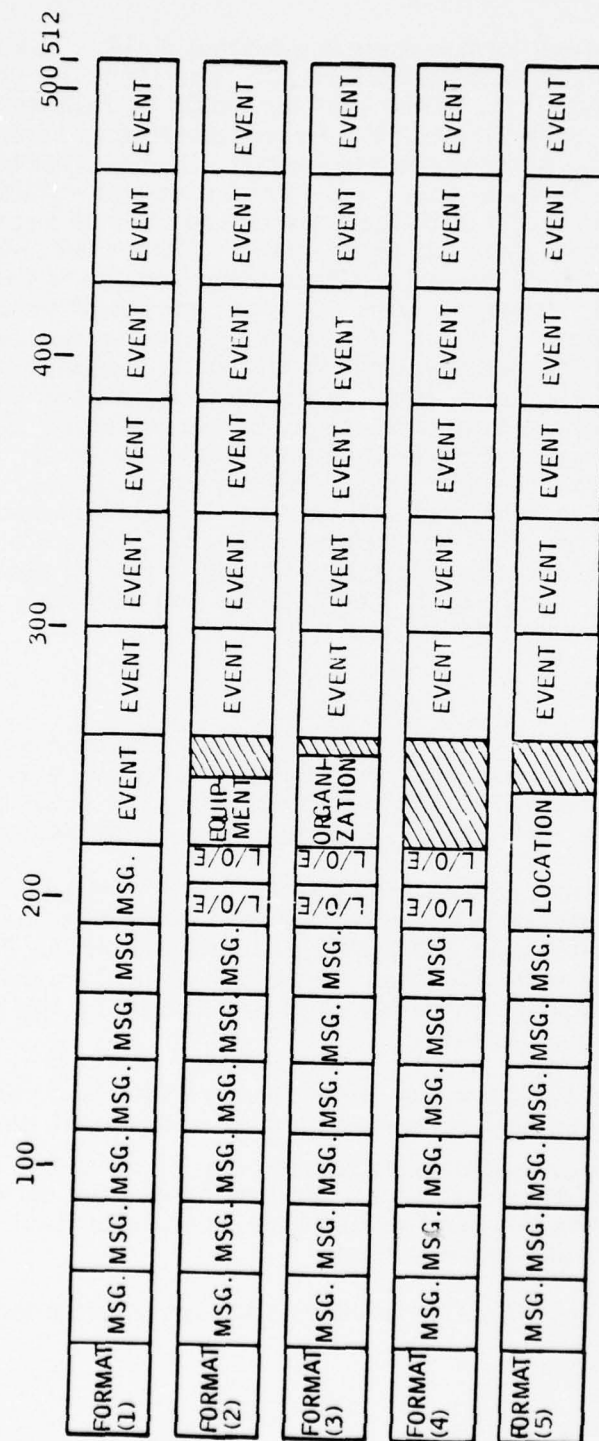


Figure 97. Formats 1 through 5 Consolidated

There are various ways to structure the format field. The conceptually simplest is to assign to each format an ID. The ID would be recorded in a subfield of the format field. This subfield would be followed by a subfield which is a busy/not busy bit vector. Corresponding to each bit in the bit vector there would be a record in the format. Hence, the bit vector would declare which records were occupied. The bit vector would be large enough to accommodate the format with the greatest number of records. The remaining space would be reserved for work space. Figure 98 shows the format fields for the five defined formats. This arrangement of the format field is not easily examined, because a given record type/position combination is not necessarily in the same column. Figure 99 shows an expanded bit vector which includes a bit for each record type/position regardless of whether it is included in the format. A further elaboration is shown in Figure 100. In this concept, two bits are used for each record type/position. One bit indicates whether the record type/position is included in the format, and the other indicates whether the space is occupied. The format field layout shown in Figure 100 is easily manipulated in software. It uses more bits, but even in a fairly large system the number of bits required would not be excessive. In theory, the format ID could be eliminated, since the bit pattern would define the ID. In practice, an ID field is convenient and may be worth the space.

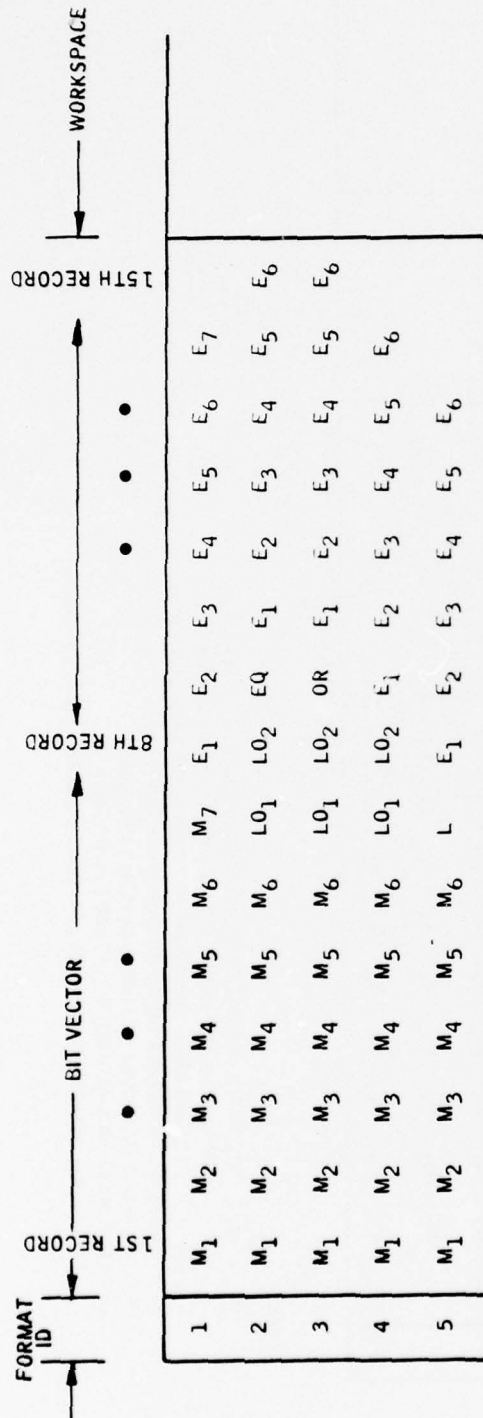
5.1.2 Data Base Size

The design objective for the ECAM has been to develop a system capable of holding 80 million or more bytes within the associative array. In developing the original estimates, allowances were included for system growth, waste space, and identification (format) information as well as actual data content.

Table 48 details the use of ECAM words for the SACWARDANS directories data base. The values shown assume that the sizes of the directories are the same as previous estimates and that the formats shown in Figure 97 are used. The total space requirement of approximately 41 million bytes is considerably lower than the design objective of 80 million bytes for the ECAM. There has been a reduction because:

- 1) The longer ECAM words provide more efficient packing and consequent reduction in waste space and format information.
- 2) Changes in the pointer structure (e. g. , use of IDs to provide access to disk records and elimination of hard disk addresses in the directories) and some other data items have resulted in some data volume reduction.

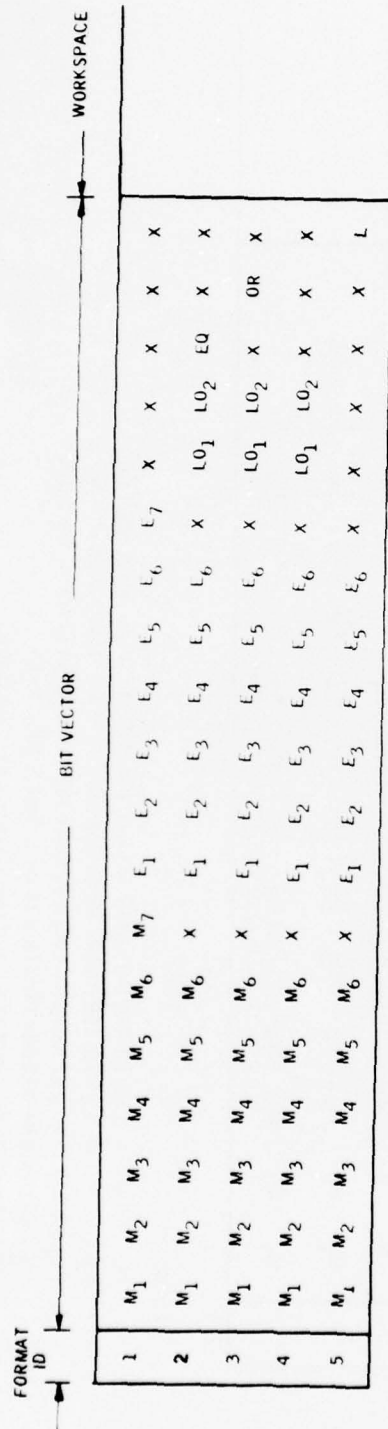
In addition, the design objective of 80 million bytes included a provision for system growth.



LEGEND:

M₁ . . . M₇ = 1ST TO 7TH MESSAGE RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 E₁ . . . E₇ = 1ST TO 7TH EVENT RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 L₀₁ AND L₀₂ = 1ST AND 2ND L/O/E RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 EQ = EQUIPMENT RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 OR = ORGANIZATION RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 L = LOCATION RECORD POSITION IS OR IS NOT 1 OR 0 BUSY

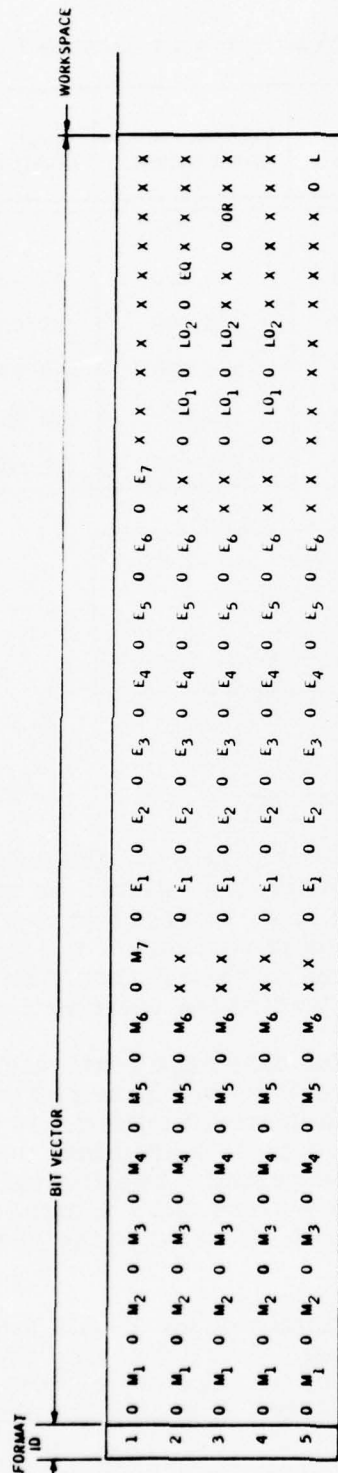
Figure 98. First Sample Format Field



LEGEND:

- M₁ . . . M₇ = 1ST TO 7TH MESSAGE RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
- E₁ . . . E₇ = 1ST TO 7TH EVENT RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
- LO₁ AND LO₂ = 1ST AND 2ND L/O/E RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
- EQ = EQUIPMENT RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
- OR = ORGANIZATION RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
- L = LOCATION RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
- X = THIS RECORD POSITION NOT PRESENT IN FORMAT (DETERMINED BY FORMAT ID)

Figure 99. Second Sample Format Field



LEGEND:

$M_1 \dots M_7$ = 1ST TO 7TH MESSAGE RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 $E_1 \dots E_7$ = 1ST TO 7TH EVENT RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 LO_1 AND LO_2 = 1ST AND 2ND L/O/E RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 EQ = EQUIPMENT RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 OR = ORGANIZATION RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 L = LOCATION RECORD POSITION IS OR IS NOT 1 OR 0 BUSY
 X = THIS RECORD POSITION NOT PRESENT IN FORMAT
 0 = THIS RECORD POSITION IS PRESENT IN FORMAT

Figure 100. Third Sample Format Field

Table 48. Use of ECAM Bytes in Sample Formats

Format	Number of ECAM Words (512 bytes/word)	Data Bytes (8-bit bytes)	Format Bytes (8-bit bytes)	Empty Bytes (8-bit bytes)	Total Bytes (8-bit bytes)
1	60,000	28,560,000	2,160,000	0	30,720,000
2	1,000	461,000	36,000	15,000	512,000
3	5,000	2,350,000	180,000	30,000	2,560,000
4	9,000	3,906,000	324,000	378,000	4,608,000
5	5,000	2,290,000	180,000	90,000	2,560,000
Total	80,000	37,567,000	2,880,000	513,000	40,960,000

% Total, data bytes	91.7
% Total, format bytes	7.0
% Total, empty bytes	1.2
% Total, non-empty bytes	98.7

5.1.3 Checkpoint, Restart, and Recovery

The system (hardware and software) must provide absolute assurance that there will be no loss of data in the event of a failure. It must also be possible to fully or partially restore the data base, subsequent to a system failure, in the ECAM memory array or in available portions of the memory array. If a portion of the array becomes inoperable, then it should be possible to use the remaining portions of the memory, loading the more critical data into it.

The most obvious way of providing for data base restoration is to periodically prepare a checkpoint copy of the directory data base and maintain a journal of changes -- or additions, deletions, and modifications. In the event of a memory failure, the last checkpoint copy is loaded into the ECAM, after which the entries in the journal are reposted. The journal is, of course, segmented into a precheckpoint copy journal and a postcheckpoint copy journal. Hence, only those entries not reflected in the checkpoint copy are reposted.

Full memory checkpoint creates an image of the ECAM memory array on another storage medium (disk or tape). A full memory checkpoint copy is, in effect, an instantaneous snapshot of the memory content. In reality, the snapshot is not prepared instantaneously. If during the time period in which

the checkpoint image is being created the content of the memory array is not altered, then the image is effectively an instantaneous snapshot. Thus, the dump operation must be undivided with respect to ECAM memory write, although it need not be undivided with respect to ECAM search and read.

Preparing a full memory checkpoint copy and restoring from it requires bulk unloading and loading of the memory. If the memory is serially dumped, a word at a time, at a shift rate of 1 million bits per second, then approximately 11 minutes are required for dumping 80 million bytes. Memory restoration would require a similar load time plus the time required for reposting the journal entries. A faster parallel I/O is provided, but it increases costs and system complexity. Hence, alternative checkpoint/restart strategies have been considered.

In a memory which is addressable, it is possible to segment a checkpoint operation. This is done by making a copy of a predefined section of the memory. Writes are locked out of that section while the copy is being made. The copy operation then moves to another section. It is always possible, when writing into the memory, to determine whether the write is to a section which has already been copied, is in the process of being copied, or has not yet been copied. Writes to the section being copied are delayed -- that is, locked out. Writes to copied sections are logged in the new journal, whereas writes to the as yet uncopied sections are logged in the old journal.

The ECAM does not have a word addressing structure that is available to the software. Hence, when an entry is added to the ECAM memory, there is no way of determining where it is placed. As a result, if the memory output is altered during a memory dump, one cannot determine whether the change is or is not reflected in the checkpoint copy. It would, of course, be possible to examine the checkpoint copy, but this would require searching it. This search would not be in the ECAM; hence, it is expensive to implement. As a result, any changes made during the checkpoint operation would have to be recorded in the journal, but it would not be clear whether the changes are or are not reflected in the checkpoint copy. If a restart is required, then reposting of all entries in the journal could result in some changes being made twice. For example, an entry could be added and included in the checkpoint copy. If the checkpoint is used for a restart, then the entry would be reposted. The ambiguities in the journal entries can probably be resolved in software procedures used in reloading the memory. Reposting can be avoided by searching the memory after the checkpoint copy has been reloaded and before the journalled entry is added. While such software procedures could be developed, they add to the complexity of the software and great care must be taken to absolutely ensure data integrity. Thus, this is not an ideal solution to the problem and it does not aid in context switching as is the case with the incremental checkpoint procedure described below.

An alternative strategy for creating checkpoint images has been developed. The alternative strategy, referred to here as incremental checkpointing, creates a total ECAM memory image as a set of smaller images, each of which is an image of a portion of the ECAM memory. The images of the ECAM memory increments cover the entire ECAM memory and they are disjoint with respect to each other. It is not, however, the case that the full image thus created is an effective instantaneous snapshot of the memory content. Each image increment is within itself an instantaneous snapshot, but a composite of two or more image increments is not an instantaneous snapshot. A procedure has been devised which permits determination of whether an update is to an area of the memory which has already been copied for the checkpoint copy. Thus, all entries in the journal are properly identified with respect to whether they are or are not included on the checkpoint copy.

The incremental checkpoint procedure is based on a software addressing scheme which is superimposed onto the ECAM architecture. The existence of the software addresses provides for determination of where new entries are placed in the memory array. Hence, it is possible to determine whether a new entry has or has not been posted to a portion of the memory array which has already been checkpointed. If the checkpoint copy has been made, then the entry is recorded in the journal which contains changes to the new checkpoint image. Otherwise, the entry is recorded in the journal only for the prior checkpoint.

The memory is subdivided into sections. Each section is assigned, probably by software, an address. The address is recorded in each word of the section. It is thus possible to locate all of the words comprising a section by searching the memory for the section address. Likewise, it is possible to determine what section an entry is in by reading the section address recorded in the word.

Checkpointing is done a section at a time. The procedure may be interrupted between sections, with array content alteration being a permissible operation. Checkpoint of a section requires a memory search on section address followed by a readout of memory contents for those words identified in the search. See Figure 101 for a flowchart.

Whenever a change is made to the array content (add entry, delete entry, modify entry), the section address is read so as to record the section address in the journal. If checkpointing is in progress, the section address is checked against a list of those which have already been checkpointed. This is used to determine whether the change should be recorded in the new journal. It should be recorded in the journal for the previous checkpoint image in either case.

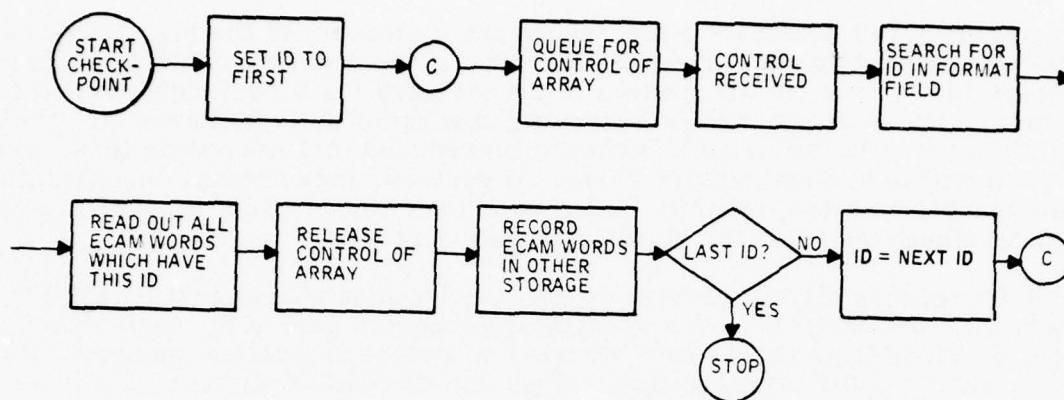


Figure 101. Checkpoint Flowchart

It is clear that the write lockout time using the incremental checkpoint procedure can be as small as desired -- depending upon the size of the section. It is also clear that restart is conceptually simple -- reload the last full checkpoint and repost all entries in its associated change journal. However, restart time has not been shortened, except for elimination of some problems with the journal.

The size and manner of selecting the memory sections is not relevant to the checkpoint procedure. It is also not necessary for a section to be in contiguous words. However the baseline ECAM storage parts, which contain 5120 bytes in 10 words, are a convenient size for an increment. Furthermore, it would seem to be convenient to move all of the data contained by a part in the event of a failure in the part. Hence, using the ECAM part as a memory increment is attractive. There may, however, be problems associated with software assignment of addresses to parts.

The above concept can be further modified to provide for retention of a dynamically updated memory image -- the equivalent to a checkpoint copy which is always up-to-date. A mirror copy is retained on disk. Each section of the ECAM memory is assigned a disk location which is accessed by the section address. All updates to the ECAM memory are also recorded in the disk copy. In the event of a total or partial memory failure, the memory would be reloaded from the disk copy which is kept current. Hence, posting of additional journalled entries would not be necessary.

Retention of a mirror copy on disk could be used as a total replacement for memory checkpointing and journalling of changes. A more conservative and safer approach is to continue periodic, but probably less frequent, checkpointing and continue to retain a change journal. It is also possible to occasionally validate the memory contents by comparing them with the disk copy.

The procedures discussed here reduce the criticality of the high I/O bandwidth in order to create checkpoint copies of the memory content. It is not acceptable to lock out all updates to the memory for a prolonged period of time, as would occur with full memory checkpointing. Incremental checkpointing permits the write lockouts to be reduced to 1 second or less, which is well within system requirements. If desired, incremental checkpointing can operate on a low-priority background task basis. That is, when the system is otherwise idle, the checkpoint task operates.

The procedures discussed here do not provide fast reload of the total ECAM memory. The retention of a dynamically-updated disk copy would speed up reload, since it would not be necessary to repost journalled changes. However, the time for total memory reload can be held to just over 1 minute by use of the high-speed I/O facility.

5.2 SLAVE CONTROLLER FUNCTIONS

This section describes the language interface between the master and the slave and discusses the microprograms and data structures required to implement this interface in the slave.

The ECAM system must operate in at least two different modes:

- Setup
- Retrieval

Essentially all usage will be in the retrieval mode, as the setup mode is required only for reformatting ECAM words to change the ECAM allocation policy, an infrequent occurrence. Furthermore, the two modes are incompatible; one user cannot be retrieving while the system's data base is being reformatted. (Limited changes -- updating, for example -- are compatible with retrieval.) This section describes the slave controller operating in the retrieval mode. The setup mode is involved primarily with preparing the tables required in retrieval; hence, no explicit discussion is included.

First, the way in which retrieval requests from analysts at terminals are handled is reviewed. After an analyst at a terminal enters a request, the following events occur:

- 1) The host software detects that a retrieval request for the ECAM has been entered.
- 2) A buffer containing the request is passed to the master machine through the host's data channel.
- 3) The master's software massages the retrieval request to produce a command sequence and appropriate data structures for the slave unit. The master's software also assigns a process number for managing the use of the slave and other resources, including space within the master's memory.
- 4) The master schedules the request and initiates slave processing when the slave becomes free.
- 5) The slave executes the sequence of commands contained in the master's memory. Some of these commands are control functions, while others request array operations. All array operands are specified as fields within ECAM records. The slave unit consults tables within the master's memory to find all instances of the desired records and fields for each operation; it issues instructions to the iteration control for the manipulation of individual fields. In this process it sequences through all instances of the record.

- 6) The iteration control performs all necessary sequencing across the bits comprising the fields within the array. The iteration control cannot make any decisions except whether the end of a field has been reached; it is not programmed.
- 7) The actual processing of the request bits is performed by the word logic associated with each word of the ECAM array.

Match bit information is kept within the ECAM words, contiguous with the record for which the bits are applicable. As described in Section 5.2.1, the match bits are processed as though they were located in a stack structure. In fact, however, they are located in fixed positions within the record and brought into the word logic hardware when the associated record is about to be processed. When the processing of the record is complete, the match bits are stored back into the ECAM word. This detail is handled by the slave unit without explicit commands within its program (in fact, these functions are generated by the slave and passed to the iteration control, which then passes them to the array hardware).

Data being passed between the slave and the array are passed through a buffer memory (discussed in Section 4.1). This buffer is constructed in two independent parts so that operands can be buffered and passed to or from the master using one buffer unit while array operations are taking place using the other buffer unit.

While the slave's primary function is to provide sequencing through the multiple instances of records within the array, it also will perform some automatic looping that will shorten the execution times for most code sequences. Specifically, it will form loops of operations wherever a sequence of operations uses the same records; the loops will minimize word shifting and match bit reloading.

In this section, the functions to be performed by the slave unit are specified. In Section 5.2.4, detailed execution sequences are presented that could be used to implement many, but not all, of the operations specified for the unit. Finally, in Section 5.2.5, the register assignment for the interpreter hardware is presented.

The instructions are described as a "high-level" language, but, in reality, almost every statement corresponds to a single instruction.

The slave controller is designed to mask from the master-produced code any need to know about multiple instances of records of the same type. Thus, code in the language is written as though there were only one instance of each record type. The instruction interpreter will implement the details required for processing multiple instances of the records.

Although the slave code automatically is generated from the retrieval request, the code will be described as though it had been produced by a programmer.

Four types of information about the language are presented in the paragraphs that follow:

- Data structures visible to the programmer
- Operations available to the programmer
- Data structures used to implement the operations (not visible to the programmer)
- Control sequences for implementing a controller which performs the desired functions.

5.2.1 Programmer Visible Data Structures

Two types of information are visible:

- Data base contents
- Intermediate processing results

The data base contents are shared among all processes, but each process has a private copy of its own intermediate results.

The basic units of data base information are field contents. Fields are grouped into records, of which there are a fixed number (although this number is implicit within certain table structures, no part of this design makes any significant assumption regarding the actual number of distinct record types) of distinct types. All processing is grouped into blocks within which all transactions interact with the same record type. (These blocks may be nested to allow processing in different records.)

Intermediate processing results are either strings, which are stored in the master's memory, integers, also stored in the master's memory, or match bits, which are stored in the record itself. The programmer sees a stack of match bits associated with each record; all manipulations of the match bits use the top entries on this stack (which is private to the process). (The appropriate match bits will be copied into the word logic during actual processing; see Section 5.2.4.)

Figure 102 summarizes these (logical) data structures.

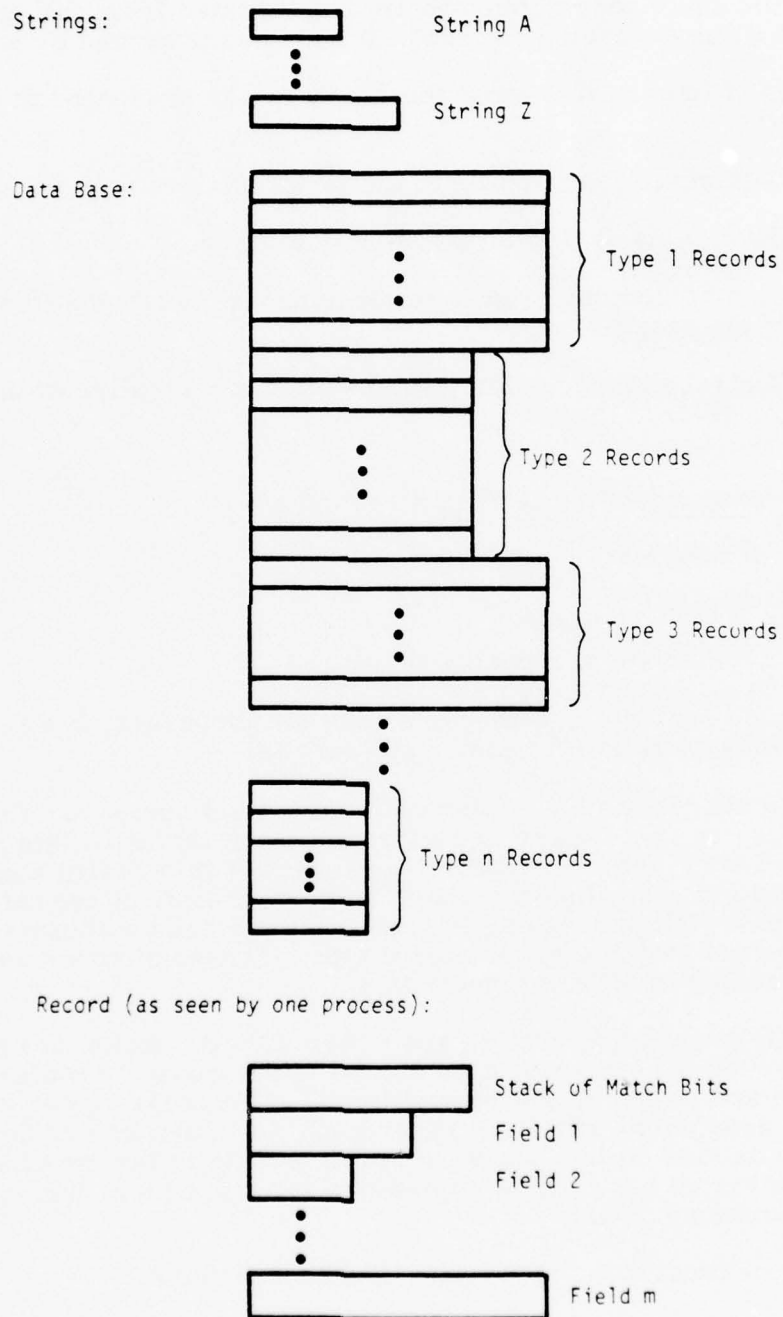


Figure 102. Logical Data Structures

5.2.2 Operations Available to the Programmer

The operations available to the programmer comprise the functional capabilities of the slave processor. Thus, in essence, this section is a programmer's reference manual for the slave processor. Most of the operations will be described by specifying a syntax (useful only for examples, since the programs are, in fact, generated automatically from the retrieval request string) and semantics as either a sequence of statements or English text. More semantic details (specified as possible implementations) for some of the operations are given in Section 4.

Programs are modularized into procedures, exactly like procedures in any programming language. Procedures are subdivided by using the compound statement FOR which includes all operations to be performed on the single record type specified in the FOR statement. FOR blocks may be nested to create more-complex sequencing among record types. The components of each FOR block are either other FOR statements or "simple" statements.

5.2.2.1 Block Structuring Statements -- The PROCEDURE statement is the only statement creating a block. A procedure block has the syntax:

```
"LABEL":      PROCEDURE ("PARAMETER_LIST");
               "DECLARATION_LIST"
               "STATEMENT_LIST"
               END;
```

(Quotation marks are used to enclose names of syntactic types.)

Control is passed to the procedure by execution of a CALL statement (see Section 5.2.2.4) in which the entry point name matches the label on the procedure block. Parameters are passed (always by reference) in the usual fashion. Any variable which can be declared can be passed as a parameter.

The PROCEDURE statement does not correspond to any code executed by the slave processor.

Any variables local to the block, including the parameters, are declared in the DECLARATION_LIST of DECLARE statements. Each declare statement has the syntax:

```
DECLARE "NAME_LIST" "TYPE" "ALLOCATION";
```

The allowed TYPES include:

- | | |
|-----------|-----------|
| • STRING | • INTEGER |
| • BOOLEAN | • FIELD |

A FIELD variable contains the name of a field. This name includes a specification of the record type which contains the field.

The ALLOCATION specifies the length of the variable; if MASTER is missing, the space is allocated in the activation block for this procedure (see Section 5.2.3.3). Otherwise, ALLOCATION specifies that the master computer holds the information:

"ALLOCATION" ::= [MASTER] "LENGTH"

Note: Declaration statements are included to assist understanding, but may not be used anywhere in the system. The allocations within the master will be performed directly by the master's request processing software and by pointer manipulations programmed by that processor. One consequence is that the LENGTHS declared in the ALLOCATION for the MASTER are not used in the processing, but do clarify examples. These lengths do appear in data structures used by the instruction interpreter (see Section 5.2.3.3), but those data structures are established by the master's request processing software. The lengths of activation block allocations are used during the prologue of the procedure to allocate space on the activation block stack (see Section 5.2.2.3.2); this is handled by a specific ALLOCATE statement (see Section 5.2.2.3.2).

The STATEMENT_LIST specifies the algorithm used in the procedure; it is constructed from compound and simple statements.

5.2.2.2 Compound Statements -- The compound statement FOR specifies iterations within records:

```
FOR { ALL
      MATCHED } RECORDS ("FIELD_DESIGNATOR") [LOCK AND] DO;
    "STATEMENT_LIST"
END_FOR;
```

The FOR statement specifies that the STATEMENT_LIST shall be executed for every instance of the record containing the designated field. (Note: In succeeding descriptions, the records containing the field designated in a FOR statement will be called the selected or designated records.) The optional LOCK action, if present, forces the selected record to be locked before entering the block; it will be unlocked upon completion of the block. If the record cannot be locked, the process will WAIT until the LOCK can be set. The match bit at the top of the match bit stack associated with the designated records is used to select which of the designated records actually participate in the processing; when MATCHED is specified, the bit present atop the stack before entering the FOR statement is used, but when ALL is specified, a "1" bit is pushed onto the stack as part of the FOR execution so that all copies of the record will participate in the initial processing steps.

FOR loops are terminated by the operator END_FOR which tests for loop completion:

END_FOR;

When all designated records have been processed, the END_FOR causes the records, if locked, to be unlocked, and the surrounding block to be reentered (which may implicitly cause a new set of records to become the designated records).

5.2.2.3 Simple Statements -- There are two types of simple statements: iterative and singular. Singular statements are control statements or require interactions between the instances of the selected record type. Iterative statements require independent operations on all instances of the selected record type.

"VALUE" is used in the following to denote a numeric quantity stored as a double-precision quantity in the master's memory, whereas "STRING" denotes an arbitrary character sequence from the master's memory. The actual addressing mechanisms and specifications used in instructions are described in Section 5.2.3.

5.2.2.3.1 Iterative Operations -- The iterative operations require independent operations on all instances of a record.

Let M_1 , M_2 , M_3 denote the three top match bits on the stack associated with the selected record. The iterative operations are:

- FIND:

Syntax: FIND ("STRING", "FIELD")

Action: $M_1 := M_1 (C(FIELD) = C(STRING))$

- MASKED FIND:

Syntax: MASKED_FIND ("STRING", "MASK", "FIELD")

Action: Same as FIND, except don't compare in bit positions where the MASK bit string contains "0".

- GREATER:

Syntax: GREATER ("STRING", "FIELD")

Action: $M_1 := M_1 (C(FIELD) .GT. C(STRING))$

- LESS:

Syntax: LESS ("STRING", "FIELD")

Action: $M_1 := M_1 (C(FIELD) . LT. C(STRING))$

- BETWEEN LIMITS:

Syntax: BETWEEN_LIMITS ("STRING₁", "STRING₂", "FIELD")

Action: $M_1 := M_1 (C(FIELD) . GE. C(STRING_1))$
 $(C(FIELD) . LE. C(STRING_2))$

- ADD:

Syntax: ADD ("STRING", "FIELD")

Action: FIELD := C(FIELD) + C(STRING)

All arithmetic FIELD operations push one bit onto the match bit stack; this bit will be set wherever an overflow has occurred. The push operation is performed in all records, but the addition and bit setting is performed only on those records where $M_1 = 1$.

- SUBTRACT VALUE:

Syntax: SUBTRACT_VALUE ("STRING", "FIELD")

Action: FIELD := C(FIELD) - C(STRING)

All arithmetic FIELD operations push one bit onto the match bit stack; this bit will be set wherever an overflow has occurred. The push operation is performed in all records, but the subtraction and bit setting is performed only in those records where $M_1 = 1$.

- SUBTRACT FIELD:

Syntax: SUBTRACT_FIELD ("STRING", "FIELD")

Action: FIELD := C(STRING) - C(FIELD)

All arithmetic FIELD operations push one bit onto the match bit stack; this bit will be set wherever an overflow has occurred. The push operation is performed in all records, but the subtraction and bit setting is performed only in those records where $M_1 = 1$.

- INCREMENT:

Syntax: INCREMENT ("FIELD")

Action: FIELD := C(FIELD) + 1

All arithmetic FIELD operations push one bit onto the match bit stack; this bit will be set wherever an overflow has occurred. The push operation is performed in all records, but the incrementation and bit setting is performed only in those records where $M_1 = 1$.

- DECREMENT:

Syntax: DECREMENT ("FIELD")

Action: FIELD := C(FIELD) - 1

All arithmetic FIELD operations push one bit onto the match bit stack; this bit will be set wherever an overflow has occurred. The push operation is performed in all records, but the decrementation and bit setting is performed only in those records where $M_1 = 1$.

- FASTREAD:

Syntax: FASTREADMATCHEDRECORDS
("CONTROL_WORD", "FIELD")

The control word contains bits for the initial state of the controller of array-host communication; generally, it specifies a BUFFER allocated in the host's memory; the separate FIELDS are concatenated to fill the BUFFER. Further details are not provided at this stage of the design.

- FASTWRITE:

Syntax: FASTWRITEMATCHEDRECORDS
("CONTROL_WORD", "FIELD")

This performs the inverse of FASTREAD; the BUFFER specified in the CONTROL_WORD is split apart inversely to the way it was concatenated in FASTREAD. The write is performed in all records having $M_1 = 1$. Further details are not provided at this stage of the design.

- PUSH1:

Syntax: PUSH1

This pushes a new match bit equal to 1 onto the stack associated with each instance of the selected record.

- PUSH0:
 Syntax: PUSH0
 Place a new cleared match bit on each stack associated with the selected record.
- POP:
 Syntax: POP
 Remove one match bit from each stack associated with the selected record.
- DUPLICATE:
 Syntax: DUPLICATE
 Push an extra copy of the top match bit onto each stack associated with the selected record.
- EXCHANGE:
 Syntax: EXCHANGE
 Action: $M_1 := M_2; M_2 := M_1$ simultaneously
 The operation is performed on all stacks associated with the selected record.
- ROTATE3PLACES:
 Syntax: ROTATE3PLACES
 Action: $M_1 := M_2; M_2 := M_3; M_3 := M_1$ simultaneously
 The operation is performed on all stacks associated with the selected record.
- NOT:
 Syntax: NOT
 Action: $M_1 := \neg M_1$
 The operation is performed on all stacks associated with the selected record.

- OR:

Syntax: OR

Action: $M_2 := M_1 \vee M_2$
POP

The operation is performed on all stacks associated with the selected record; the stack is now one bit shorter.

- AND:

Syntax: AND

Action: $M_2 := M_1 \cdot M_2$
POP

The operation is performed on all stacks associated with the selected record; the stack is now one bit shorter.

- XOR:

Syntax: XOR

Action: $M_2 := M_1 \oplus M_2$
POP

The operation is performed on all stacks associated with the selected record; the stack is now one bit shorter.

- READ:

Syntax: READ ("VALUE")

Action: PUSH
 $M_1 := M_{\text{VALUE}+1}$

The operation is performed on all stacks associated with the selected record; the stack is now one bit longer. This operation reads a bit from a lower position on the match bit stack and copies it to a new top of the stack entry.

- WRITE:

Syntax: WRITE ("VALUE")

Action: $M_{\text{VALUE}} := M_1$
POP

The operation is performed on all stacks associated with the selected record; the stack is now one bit shorter. This operation writes the match bit value from the top position on the stack to a position within the stack and then pops the value off the stack.

5.2.2.3.2 Singular Operations -- There are three classes of singular operations:

- Control operations
- Manipulation operations applied to values in the master
- ECAM operations requiring interactions between selected records.

5.2.2.3.2.1 Control Operations -- Generally, the control operations affect the execution sequence within the slave. However, some control operations actually manipulate the implementation data structures used by the slave. Hence, we must describe the stack of activation blocks.

A stack of activation blocks associated with each process is stored within the master's memory. There is a block on this stack for each procedure which has been called by the process but which has not returned control to its caller. Each block has the format shown in Figure 103. The elements within the block

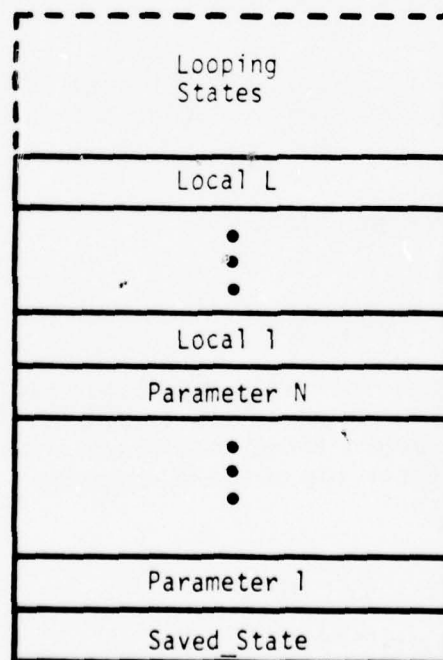


Figure 103. Activation Block Format

are described in Section 5.2.3.3. Here we only need to know that the saved state at the bottom of the block contains two words: one contains a pointer to the next instruction in the calling program and the other contains a pointer to the bottom of the preceding block on the stack.

The control operations are:

- GO TO:

Syntax: GO_TO ("ADDRESS")

Action: Take the next slave instruction from the designated address (within the master's memory).

- IF POSITIVE GO TO:

Syntax: IF_POSITIVE_GO_TO ("ADDRESS")

Action: If the value on the top of the activation block stack is positive, take the next instruction from the designated address. Otherwise, continue with normal instruction sequencing. See Section 5.2.2.3.2.1 for instructions which place values at the top of the stack where they may be tested by this instruction. The result being tested is popped from the stack.

Note: This statement can be used to create a loop within the program, as shown in the following code skeleton:

```
FOR.....DO;  
    GO_TO (LABEL2);  
LABEL1:  
    (body of loop)  
LABEL2: (evaluate condition)  
    IF_POSITIVE_GO_TO (LABEL1);  
END_FOR;
```

- SAVE BLOCK:

Syntax: SAVE_BLOCK

Action: Allocate two words on the activation block stack and place a pointer to the present activation block in the first word of the pair. Keep a pointer to the pair for use during the CALL instruction.

Note: This description precludes the use of subroutine calls between SAVE_BLOCK and CALL to evaluate parameters of the procedure being called. If this possibility is desirable, simply save the pointer to the SAVE_BLOCK word and include a flag to indicate which calls have been made -- the descriptions of CALL and RETURN would require corresponding changes.

- CALL:

Syntax: CALL "ADDRESS"

Action: Call the procedure whose first instruction is located at the designated address. This requires saving the instruction counter in the SAVE_BLOCK word which is pointed to by the saved pointer (see SAVE_BLOCK description above), and updating the activation block pointer (see Section 5.2.3.3.2).

- ALLOCATE:

Syntax: ALLOCATE "VALUE"

Action: Allocate VALUE new locations at the top of the activation block stack. The new locations will be used to hold the local variables for the new block.

- RETURN:

Syntax: RETURN

Action: Return from the current procedure to the calling procedure. This requires freeing the topmost activation block on the activation block stack and making corresponding changes in the slave's registers describing the status of the activation block.

- HALT:

Syntax: HALT

Action: Terminate this process; signal the master that the process has terminated and the slave is now free for a new assignment.

- WAIT:

Syntax: WAIT

Action: Suspend this process; signal the master that the process has been suspended and the slave is free for another assignment. The process must communicate information to the master regarding the conditions for resuming the processing. Note that the master may not reassign any resources allocated to the process after WAIT, whereas all resources must be freed after HALT. These actions are taken by the master's control program.

5.2.2.3.2.2 Value Manipulation Operations -- All of these operations manipulate double-precision quantities stored within the master's memory. Any address which is zero will be interpreted to refer to the top (double word) entry of the activation block stack: if the operand is read in the operation, the stack will be popped; if written, a new top entry will be pushed onto the stack. Within the syntax, the reserved word STACK is used to designate the top of the stack used in this way.

The value manipulation operations are:

- ADD:

Syntax: ADD "VALUE₁" TO "VALUE₂" GIVING "VALUE₃"

Action: Perform double-precision addition.

- SUBTRACT:

Syntax: SUBTRACT "VALUE₁" FROM "VALUE₂" GIVING "VALUE₃"

Action: Double-precision subtraction.

- COPY:

Syntax: COPY "VALUE₁" TO "VALUE₂"

Action: Copy a double word. Note that by using STACK operand addresses, PUSH and POP actions can be achieved on the activation block stack.

- STRING COPY:

Syntax: STRING_COPY "STRING₁" TO "STRING₂", "LENGTH"

Action: Length specifies the number of words to be copied, while STRING₁ and STRING₂ are starting addresses.

- ADDRESS COPY:

Syntax: ADDRESS_COPY "ADDRESS₁" TO "ADDRESS₂"

Action: Place a copy of the first address in the location specified by the second address. The first address will be converted to absolute form before being copied.

Note: This operation will usually be used to place the address of a subroutine's parameter on the activation block stack.

- MAKE POINTER:

Syntax: MAKE_POINTER "INDEX"

Action: INDEX is an index into the current activation block. Convert this to a memory address by adding it to the pointer to the current activation block; push the result onto the top of the activation block stack.

5.2.2.3.2.3 Record Interaction Operations -- The operations in this class use the values in selected fields from possibly all instances of the record in the ECAM.

The record interaction operations are:

- SELECT FIRST:

Syntax: SELECT_FIRST

Action: Push a new set of match bits onto the match bit stacks associated with the designated record; all new bits are cleared except the one on the first stack which had $M_1 = 1$ before the SELECT_FIRST instruction was executed. Note that the ordering implied by the word "first" is determined by the wiring of the multiple match resolver. This ordering is arbitrary, but deterministic.

- READ FIRST:

Syntax: READ_FIRST ("STRING", "FIELD")

Action: Copy the contents of the designated field in the first selected ($M_1 = 1$) record to the master's memory. The first record is determined by the multiple match resolver. The match bits are unchanged.

- WRITE FIRST:

Syntax: WRITE_FIRST ("STRING", "FIELD")

Action: Copy STRING into the designated field within the first selected record. The match bits are unchanged.

- RESET FIRST:

Syntax: RESET_FIRST

Action: Clear the first set match bit atop the stacks associated with the designated record. All other match bits remain unchanged.

- READ FIRST ELIMINATING DUPLICATES:

Syntax: READ_FIRST_ELIMINATING_DUPLICATES ("STRING", "FIELD")

Action: Read the designated field from the first selected record; reset its match bit (M_1), and reset the match bits of all other selected records whose contents in the designated field match the string read out to the master's memory.

- MATCH COUNT TO:

Syntax: MATCH_COUNT_TO "VALUE"

Action: Place the count of set M_1 match bits into VALUE, which is a double-precision integer in the master's memory. Do not modify any match bits.

- STATUS TO:

Syntax: STATUS_TO "STRING"

Action: Copy the status of the ECAM system to the specified string within the master's memory. The detailed structure of the status information is not defined in this document.

- MAXIMUM:

Syntax: MAXIMUM ("FIELD")

Action: Consider all selected records with $M_1 = 1$. Clear M_1 in all records in which the contents of the designated field are not equal to the maximum value among all fields being considered. The result will be $M_1 = 1$ only where the maximum value is found.

- MINIMUM:

Syntax: MINIMUM ("FIELD")

Action: Consider all selected records with $M_1 = 1$. Clear M_1 in all records in which the contents of the designated field are not equal to the minimum value among all fields being considered. The result will be $M_1 = 1$ only where the minimum value is found.

5.2.2.3.3 Notes -- Every iterative operation must be iterated across all instances of the designated record. Since these records do not interact in the performance of the operation, the order of iteration does not affect the results. Suppose that a program fragment contains a sequence of iterative operations:

OP1

OP2

OP3

Suppose there are four instances of the record: R_1, R_2, R_3, R_4 . Note that these represent instances within formatted ECAM words, and that the instances may in fact be in words of different formats; the record instance table (see Section 5.2.3.1) specifies these details. A simple sequencing of the operations through the records is $OP1(R_1), OP1(R_2), OP1(R_3), OP1(R_4), OP2(R_1), \dots OP3(R_4)$. However, the same results will be achieved by any sequence which includes all combinations of the operations and record instances, as long as the operations on each record instance are performed in the specified order. The sequence $OP1(R_1), OP2(R_1), OP3(R_1), OP1(R_2), \dots OP3(R_4)$ is

equally valid; it is constructed by forming a loop around the sequence and executing the loop for each instance of the designated record. Loops of this type may contain any sequence of iterative operations, but they may not contain singular operations or FOR statements.

The slave will automatically form these loops whenever possible; they can be efficient because they minimize the need to load and store the match bits within the array.

5.2.2.4 Example -- Here is a procedure which searches for and marks all records in which field D contains string W, or field C matches the contents of field B in some other record (of possibly another type) in which field A contains string Q.

```
MARK: PROCEDURE (A, B, C, D, Q, W);
      DECLARE A, B, C, D FIELD;
      DECLARE Q, W, CONTENTS STRING;
      ALLOCATE CONTENTSLENGTH;
      FOR MATCHED RECORDS (D) DO;
        PUSH0;          /*INITIALIZE THE BIT WHICH WILL
        END_FOR;        COLLECT (BY OR) THE MATCH OF C
                        AGAINST THE B OF SOME RECORD*/
      FOR ALL RECORDS (A) DO /*DO THIS MATCH IN ALL
                            INSTANCES*/
        FIND (Q, A);    /*MARK ALL THAT MATCH STRING Q*/
        END_FOR;       /*THIS COULD BE REMOVED IF THE
                        NEXT FOR ALL RECORDS (A) IS
                        ALSO REMOVED*/
      FOR MATCHED RECORDS (A) DO;
        /*LOOP OVER ALL MATCHES OF Q IN A*/
        GO_TO END_SOME;
      READB_LOOP: READ_FIRST (CONTENTS, B); /*READ ONE B FIELD
        INTO CONTENTS--THIS DOES NOT
        CHANGE MATCH BITS*/
      DELETE_FIRST; /*REMOVE THE USED B FROM FURTHER
        CONSIDERATION*/
      FOR ALL RECORDS (C) DO; /*NOW LOOK FOR CONTENTS IN
        ALL INSTANCES OF C*/
        FIND(CONTENTS, C); /*MARK EVERY C THAT
        MATCHES THIS B*/
      OR;                /*COMBINE THIS RESULT WITH
        END_FOR;         PREVIOUS B SEARCHES*/
```



```

END_SOME: SUBTRACT =1 FROM MATCHCOUNT GIVING STACK;
          /*EVALUATE TERMINATION CONDITION*/
IF_POSITIVE_GO_TO READB_LOOP;
          /*LOOP IF THERE ARE ANY MORE MATCHES*/
END_FOR;
FOR ALL RECORDS (A) DO; /*COULD BE REMOVED; BUT
                        SEE THIRD END BACK*/
POP;      /*REMOVE THE BITS WHICH HOLD THE
          A MATCH RESULT*/
END;
FOR ALL RECORDS (D)
FIND(W,D); /*MARK ALL D's CONTAINING W*/
OR;        /*COMBINE THIS RESULT WITH
          PREVIOUS RESULT; THIS FORMS
          THE ROUTINE'S RESULT*/
END;
RETURN;    /*END OF EXECUTION*/
END;       /*END OF PROCEDURE DEFINITION*/

```

Notice that the record and field-naming conventions are redundant. Each field is assumed to have a unique name; thus, its record can be inferred from the field name (a rule similar to the PL/1 structure naming rule). Each compound statement contains a field designation that implicitly selects a record; each field used within the block must be within the same record. This condition should not be violated because the request processing software will generate consistent code; it will be checked, however, in the instruction interpreter, which will cause an interrupt in the master if a mismatch is detected.

5.2.3 Implementation Data Structures

The data structures described in this section are used in the implementation of the language (i. e., by the instruction interpreter), but most are not visible to the programmer of the slave machine. Most of these structures describe memory allocations for a process, for fields within records, or for records within ECAM words. These are discussed in reverse order.

5.2.3.1 Records within ECAM Words -- Each ECAM word has one of a fixed number of fixed formats; each word contains a format field to hold the format designator. In order to locate all records of the same type in words of any format, the first records are classified by type number. For each record type there is a descriptor of a list of instances of the record type, a LOCK field, and a STACKLENGTH field (see Figure 104). LOCK contains zero or

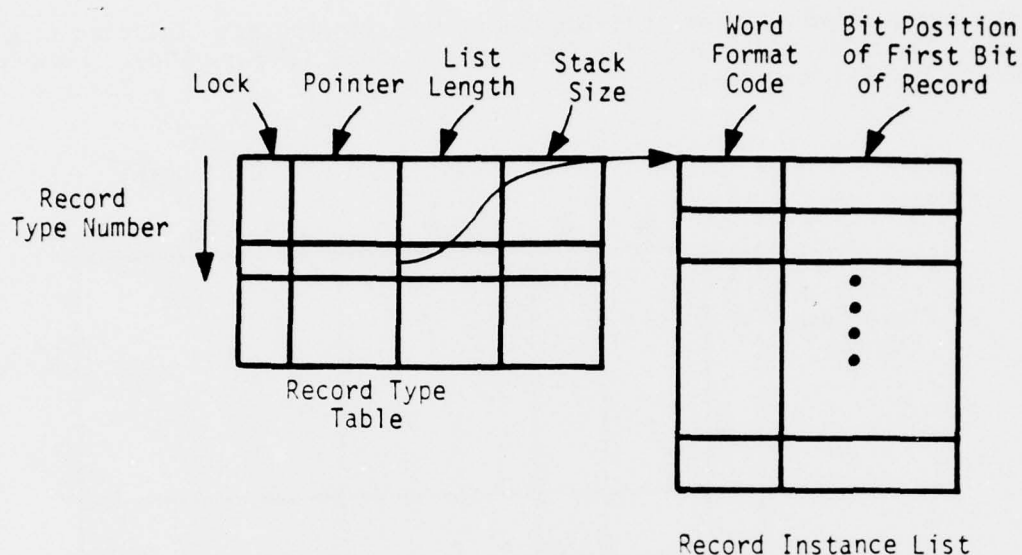


Figure 104. Record Descriptor Structures

the process number of a process which has locked this record type along with a bit pattern which indicates the allocation of the match bit stack slots within the record. (This allocation is handled by software within the master.) Finally, the STACKLENGTH field contains the maximum size of match bit stacks associated with the record (see Section 5.2.3.3).

Each descriptor of an instance of a record contains the format code and bit origin which specify the search algorithm to find an instance of the record. All instances of a selected record type can be found by looping through the corresponding record instance list until its length limit has been reached.

Note that these tables do not change (except within the LOCK fields) unless the structure of the data within ECAM is changed. All processes use these tables.

5.2.3.2 Fields within Records -- Each field can occur only once within a record. Its origin and length within the record specify the bits in the field. The record type is included in the field descriptor (see Figure 105). An order bit specifies whether the LSB or MSB comes first within the field.

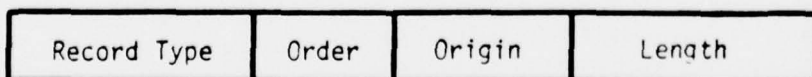


Figure 105. Field Descriptor

There are a fixed number of fields whose descriptors are collected in a field descriptor table, indexed by the field type number (Figure 106). This table is common to all processes.

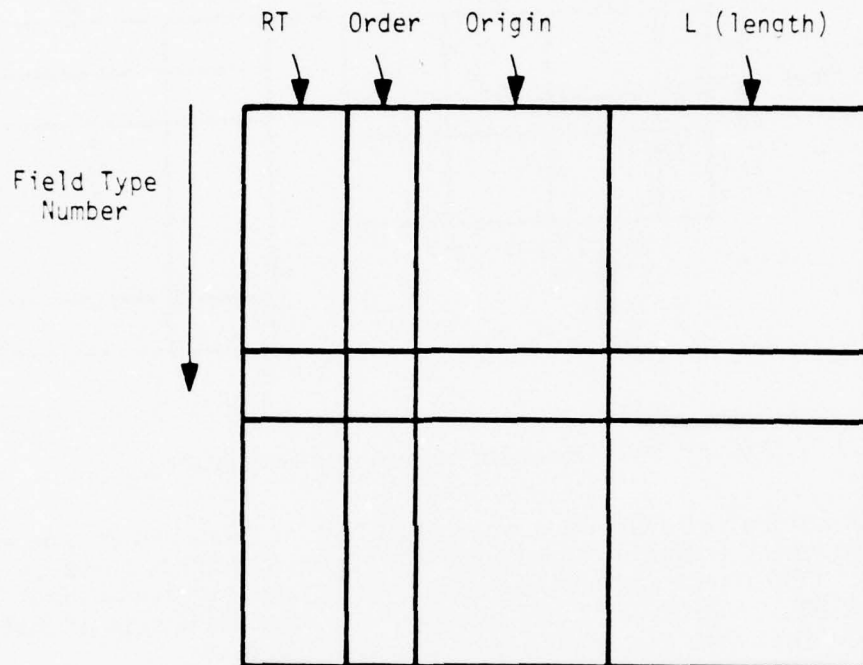


Figure 106. Field Descriptor Table

5.2.3.3 Process Local Information -- There are three types of process local information:

- Match bits in records
- Buffer spaces in the master and host
- Other miscellaneous state information

5.2.3.3.1 Match Bits In Records -- A process may have a stack of match bits in every instance of every record type. Fortunately, most processes do not require so many stacks. Each record contains a small field holding some of these match bit stacks (the space can be sized only after more information about the application is available). A table is needed for each process describing the locations of its match bit stacks for each record type (see Figure 107). The TOP describes the current length of the stack. Assume (arbitrarily) that the bottom of the stack has the smallest bit address (BASE).

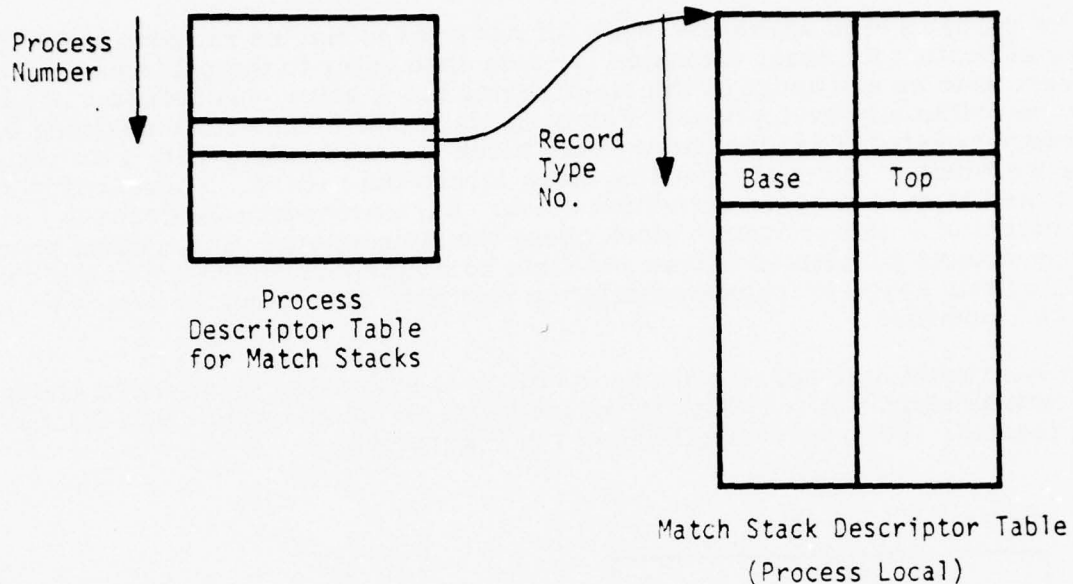


Figure 107. Match Stack

The sum of BASE and TOP will be the bit address of the top of the stack. The amount of space allocated for each stack may be different for each record; the actual allocated length will be found in the record descriptor (see Section 5.2.3.1).

5.2.3.3.2 Miscellaneous Process State -- All information associated with procedure calls is saved within a stack, grouped into activation blocks; each activation block corresponds to a single activation of some procedure. The format of an activation block (see Figure 103) is described next. The items within the block are:

- **SAVED_STATE:** The addresses used to return to the calling program upon completion of this activation. These include the saved program counter and record index and the pointer to the preceding activation block.
- **PARAMETERS:** Parameters passed to the procedure, placed in the stack by the calling procedure.
- **LOCALS:** Local variables used during the execution of this procedure, allocated space upon entry to the procedure.
- **LOOPING STATE:** The record type numbers of records specified in FOR statements whose statement lists have not been completed. These may be intermixed with numeric results pushed on the stack by arithmetic operations.

The space in each activation block for declared variables remains fixed after the allocation for local variables is made upon entry to the procedure. However, entries are added to the stack during block entry, procedure call, and some arithmetic evaluations. For each block entry one word containing the record type index for the surrounding block is saved, along with a bit indicating whether the corresponding block locked the record. These words are stacked at the top of the activation block. Each procedure call causes the creation of a new activation block above the current one. The calling procedure inserts parameter values and state holding space on the top of the current block; this space is included within the next activation block by action of the CALL operator.

An item within the current activation block is addressed by an index giving its location relative to the "bottom" of the block -- the end where SAVED_STATE is located. The addressing is shown in Figure 108.

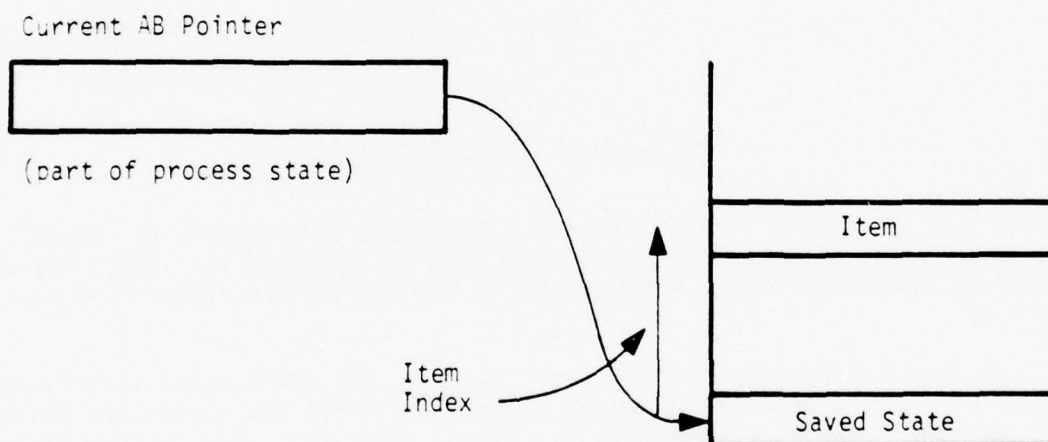


Figure 108. Addressing Within Activation Blocks

It also is necessary to access stacked items not in the current block, although this is limited to references to parameters. The addresses of such items will be specified by an appropriate master memory address.

5.2.3.4 Addressing Summary -- This design is based on the assumption that there will be no dynamic relocation within the master's memory that is seen by the slave controller.

Each address within a slave instruction contains two control bits and an address field. The context of each address (the operation code plus the position of the operand for the operation) determines whether a location within the master's memory or a table index (for a field) is expected. Figure 109 summarizes the addressing data structures.

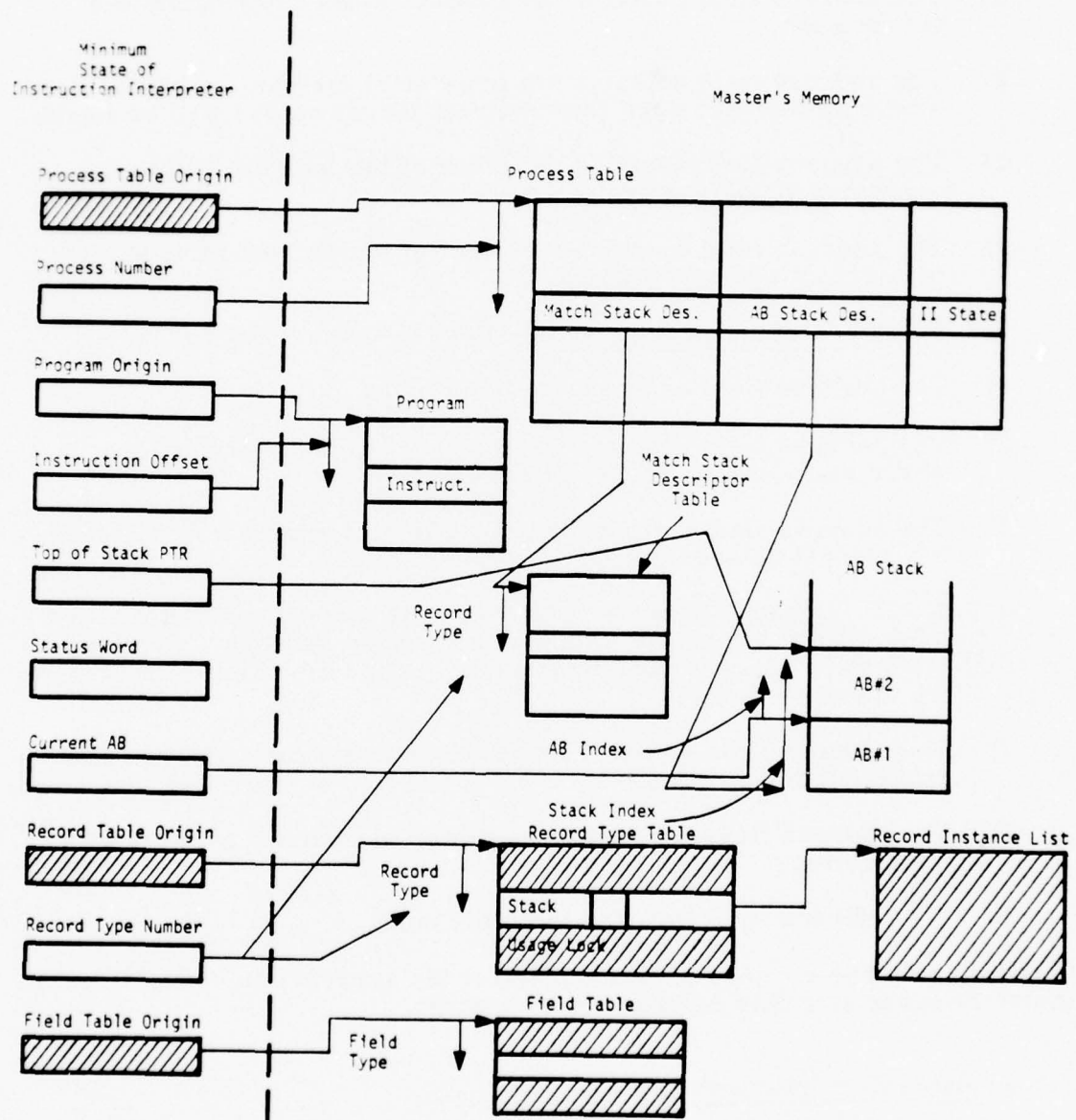


Figure 109. Addressing Data Structures

5.2.3.4.1 Location Expected -- There are four cases:

- 1) The address field contains the (master's) memory address of the location.
- 2) The address field contains the (master's) memory address where an indirect word (see Indirect Words below) will be found.
- 3) The address field contains the index of the location within the current (topmost) activation block.
- 4) The address field contains the index of an indirect word within the current activation block.

5.2.3.4.2 Table Index Expected -- Here we have three cases:

- 1) The address field contains the table index.
- 2) The address field contains the (master's) memory address where an indirect word will be found.
- 3) The address field contains the index of an indirect word within the current activation block.

5.2.3.4.3 Indirect Words -- Indirect words may specify further indirection, in which case they contain a master's memory address. Otherwise, their interpretation depends upon whether a location or table index is expected. Thus, there are three cases:

- 1) The address field contains the (master's) memory address of another indirect word.
- 2) The address field contains the (master's) memory address of the location.
- 3) The address field contains the table index.

Note that only one bit need be used to select the appropriate case, since the choice of cases 2 or 3 is determined by context.

5.2.4 Instruction Interpretation

The instruction interpreter reads instructions of the types specified in Section 5.2.2 and issues command sequences to the iteration controller for proper execution of those instructions. This section describes one way to implement many of the instructions. These specifications have been designed for reasonably efficient execution; usually, tradeoffs have been made in favor of keeping the array busy even though more references to the master's memory may be required.

Within these descriptions, PL/1 structure-like notation is used to denote fields within objects, as in OBJECT.FIELD. The PL/1 declarations of these structures will not be given, so the reader must rely on previous figures plus his intuition.

5.2.4.1 Address Generation -- Operand specifications within the implementation sequences of Section 5.2.4.3 will be used. The reader should consider each operand name to be the target address determined by following all indirection chains to find either a memory address or a field number. The necessary addressing algorithms can be generated easily from the descriptions in Section 5.2.3.4.

5.2.4.2 Iteration Controller Commands -- The following discussion summarizes the commands that the interpreter may send to the iteration controller. They are separated into four groups. The first three groups involve the word logic. They are: 1) match manipulation commands which affect the match and T bits; 2) processing (and searching) commands; and 3) I/O commands. The fourth group, slave commands, results in alteration of registers within the slave itself. The reader may wish to refer to Section 4.2 during this discussion. The microcode which maps this set of commands into array control sequences is discussed in Section 5.2.4.3.

5.2.4.2.1 Group 1. Match Manipulation Operations -- The match manipulation operations are:

- **SAVE_m(INDEX):** Copies value of m into MM_{index} (for all words); m is unchanged, T is set to 1.
- **SELECT_m(INDEX):** Copies value of MM_{index} into m (for all words); MM_i is unchanged, T is set to 1.
- **STORE_m(INDEX):** Copies value of MM_{index} into storage where $m = 1$; storage is unchanged where $m = 0$. MM_i and m are unchanged, T is set to 0, storage is advanced 1 bit position.
- **STORESTACK(INDEX):** Copies value of MM_0 through MM_{index} into INDEX+1 bits of storage where $m = 1$; storage is unchanged where $m = 0$. MM and m are unchanged, T is set to 0, storage is advanced INDEX+1 bit positions.
- **LOAD_m(INDEX):** Copies value of storage into MM_{index} where $m = 1$; MM_{index} is cleared where $m = 0$. Storage and m are unchanged, T is set to 0, storage is advanced 1 bit position.
- **LOADSTACK(INDEX):** Copies value of INDEX+1 storage bits into MM_0 through MM_{index} where $m = 1$; clears MM_0 through MM_{index} where $m = 0$. Storage and m are unchanged, T is set to 0, storage is advanced INDEX+1 bit positions.

The following Group 1 instructions involve operations on T and m and operate in a common fashion:

<u>Operation Name</u>	<u>Operation</u>
SETm	$1 \rightarrow m$
CLEARm	$0 \rightarrow m$
SETT	$1 \rightarrow T$
CLEART	$0 \rightarrow T$
COMPLEMENTm	$\bar{m} \rightarrow m$
COMPLEMENTT	$\bar{T} \rightarrow T$
MTOT	$m \rightarrow T$
TTOM	$T \rightarrow m$
mEXCHANGET	$T \leftrightarrow m$
mANDTTOm	$m, T \rightarrow m$
mORTTOM	$m+T \rightarrow m$
mXORTTOM	$m \oplus T \rightarrow m$

These operations occur in all words and have no side effects (unspecified variables are unchanged); storage is not shifted.

The following Group 1 instructions involve use of the multiple match resolver and are deterministic in that, for a given vector in m, the choice of m's to be Selected/Reset is time-invariant:

- **SELECTFIRST:** Sets all except one of currently true (=1) m bits to zero; T, MM and storage are unaffected.
- **SELECTGROUP:** Sets all except 10 of currently true m bits to zero; T, MM and storage are unaffected. (If 10 or fewer m's were true, all remain = 1.)
- **RESETFIRST:** Clears one of the set of currently true m bits; T, MM and storage are unaffected.
- **RESETGROUP:** Clears 10 from the set of currently true m bits; T, MM and storage are unaffected. (If 10 or fewer m's are true, all are reset.)
- **MATCHCOUNT:** Execution of this instruction results in a binary count of the number of m's = 1 being placed in a register available to the interpreter (see the Group 4 instruction discussion below). No state changes occur in the word logic.

5.2.4.2.2 Group 2. Processing Functions -- This group includes both searching (match) and arithmetic operations. At the interpreter level, the ECAM arithmetic processing functions occur in pairs corresponding to MSB-first or LSB-first ordering of arithmetic fields. This means that, unless MSB-first and LSB-first algorithms exist for a given function, one member of the function pair requires that bits be accessed in descending order of bit addresses. As may be recalled, the storage device baseline assumes only a unidirectional shift capability. Thus, access in the reverse order implies that 256 (the block size) + 1 shifts occur between each bit. In only one case (GREATER-THAN) do MSB-first and LSB-first algorithms exist. For the others, it is necessary to adapt to the time-cost by choosing the bit-ordering within a field to minimize total processing time.

In the discussions that follow, the function names indicate the bit (MSB/LSB) that is to be processed first. An "R" indicates that the array is to be shifted in the reverse direction. In either case, storage must be positioned at the first bit to be processed prior to beginning the function. Figure 110 shows the alternative field organization and shift direction possibilities.

The processing functions are:

- ADDLSB(LENGTH, BUFFER), ADDLSBR(LENGTH, BUFFER): Adds a number from the parameter buffer to the LENGTH-bit field in all words where $m = 1$. Words where $m = 0$ are unchanged, m and MM are unchanged, T contains the carryout of the MSB, and storage is shifted by LENGTH bits. (Note: Subtraction of the parameter from all matched words is done by complementing the parameter in the interpreter.)
- REVSUBLSB(LENGTH, BUFFER), REVSUBLSBR(LENGTH, BUFFER): Replaces the LENGTH-bit field in all words where $m = 1$ with the value in the parameter buffer minus the value previously stored in the field. Words with $m = 0$ are unchanged, m and MM are unchanged, and T is the carryout of the MSB.
- ACOMPMSB(LENGTH, BUFFER), ACOMPMSBR(LENGTH): Compares the LENGTH-bit parameter in the buffer with the corresponding field in all words where $m = 1$; produces coded result in m and T as follows:

$M, T =$	00 nonparticipant
	01 field < parameter
	10 field > parameter
	11 field = parameter

where $m = 0$, T is set to 0, and storage is unchanged. (Note 1: This operation interprets field and parameter values as positive binary integers. Where negative numbers are in the permissible

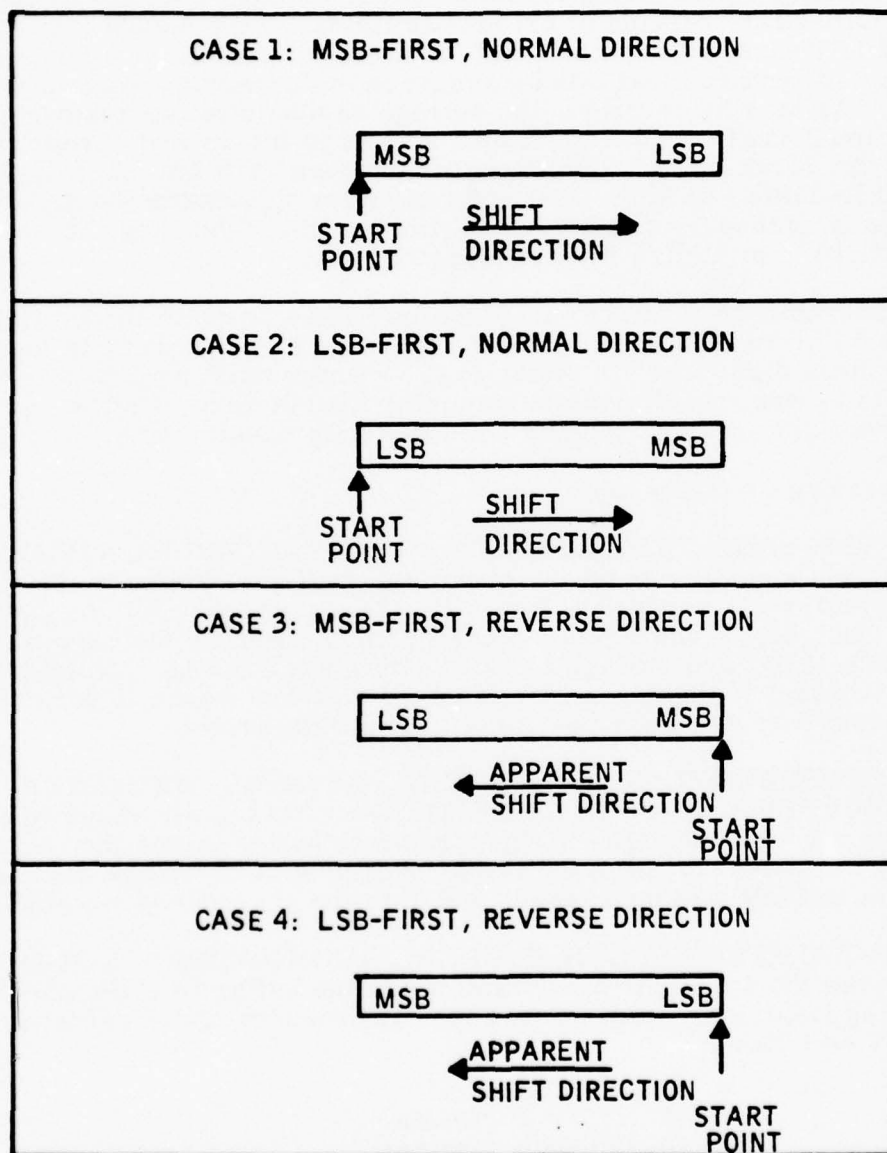


Figure 110. Alternative Field Organization and Shift Direction Possibilities

domain, the positive and negative sets must be separated prior to use of ACOMPMSB. Note 2: The equality test of this function is used for exact-match searches on non-numeric fields where masking is not required. In this case, the ACOMPMSB form is always used since field order is irrelevant.)

- MINMSB(LENGTH, BUFFER), MINMSBR(LENGTH, BUFFER) and MAXMSB(LENGTH, BUFFER), MAXMSBR(LENGTH, BUFFER): For the set of words where $m = 1$, all m 's are set to zero except those marking words which contain the maximum or minimum field values within the set. Fields are interpreted as positive integers; where m was 0, it is unchanged. Storage is not altered, but is shifted by LENGTH bits, the previous value of T is destroyed.
- COMPLEMENT(LENGTH): Replaces the LENGTH-bit field with its two's-complement in all words where $m = 1$. Words where $m = 0$ are unchanged, m and MM are unchanged, and T is set to 0. (Note: Works independently of field bit order.)
- MASKEDSEARCH(LENGTH, BUFFER): Compares the LENGTH-bit parameter with the corresponding field of all words where the second parameter in the buffer is zero, m remains set in all words containing matches, words with $m = 0$ are unchanged, T is unchanged, and storage is shifted by LENGTH bits.

5.2.4.2.3 Group 3. Input/Output Functions -- The I/O functions are:

- WRITEALL(LENGTH): Transfers the LENGTH-bit parameter from the buffer into all words where $M = 1$. Words with $m = 0$ are unchanged, m and T are unchanged, and storage is shifted by LENGTH bits.
- READALL(LENGTH): Transfers the logical-OR of the values of the LENGTH-bit fields in all words where $m = 1$ into the parameter buffer. Words with $m = 0$ do not participate, m and T are unchanged, and storage is shifted by LENGTH bits. (Note: This capability is provided in the array as a component of the MAX/MIN algorithm. It is extended to the programmer level via this function code so that it may be used if needed.)
- WRITEFIRST(LENGTH): Transfers a LENGTH-bit parameter from the buffer into the word having $m = 1$. Words having $m = 0$ are unchanged, m and T are unchanged, and storage is shifted by LENGTH bits. (Note: This operation can only be performed when there is a single m that is true. This can be guaranteed by preceding the WRITEFIRST with a SELECTFIRST operation.)

- READFIRST(LENGTH): Transfers the LENGTH-bit field from the word having $m = 1$ to the parameter buffer. Words having $m = 0$ are unchanged, m and T are unchanged, and storage is shifted by LENGTH bits. (See note for WRITEFIRST.)
- READFIRSTANDMARKDUPES(LENGTH): Operates like READFIRST, but also sets $T = 1$ in all words having field values identical to the one read out. (Note: Setting of T is independent of m and MM values.)
- READGROUP, WRITEGROUP: These operations will be implemented within the Fast I/O controller, so cannot be specified in detail at this time. In general, they will work like the READFIRST/WRITEFIRST pair, except the data source/destination will be either host memory or some system I/O device. Execution of either of these will involve the interpreter's relinquishing control of the array to the Fast I/O controller for the duration of a 1 to 10 word group transfer. Groups are chosen via the SELECTGROUP operation prior to beginning the transfer.

5.2.4.2.4 Group 4. Slave Commands -- This group of functions is the method by which the interpreter reads and writes the working registers of the interpreter. With the exception of the TO functions, they will primarily be used for maintenance and diagnostic purposes.

The first two are transmitted through the normal buffer mechanism:

- TO(BIT): Copies the value (BIT) from the Length register into the Bit Address register of the Iteration controller. This action causes storage to be advanced so that the indicated bit is ready to be operated upon.
- READREG(INDEX): Copies the contents of the indexed register into the IT Output Register.

Index codes will be provided for:

- BIT COUNT
- BIT ADDRESS
- MICROADR
- FIRST ADR
- PHYSICALADR

Registers as discussed in Section 4.1.

The third Group 4 "instruction" is access to the IT Output Register as a source on the interpreter bus. After executing a MATCHCOUNT or a READREG operation, the interpreter obtains the result by accessing the Output Register.

For reference purposes, the Iteration controller commands are summarized in Table 49.

Table 49. Iteration Controller Commands Summary

<u>Group 1:</u>	<u>Group 2:</u>
SAVEm	ADDLSB, ADDLSBR
SELECTm	REVSUBLSB, REVSUBLSBR
STOREm	ACOMPMSB, ACOMPMSBR
STORESTACK	MINMSB, MINMSBR
LOADm	MAXMSB, MAXMSBR
LOADSTACK	COMPLEMENT
SETm	MASKEDSEARCH
CLEARm	
SETT	<u>Group 3:</u>
CLEART	WRITEALL
COMPLEMENTm	READALL
COMPLEMENTT	WRITEFIRST
MTOT	READFIRST
TTOM	READFIRSTANDMARKDUPES
mEXCHANGET	READGROUP
mANDTTOm	WRITEGROUP
mORTTOM	
mXORTTOM	<u>Group 4:</u>
SELECTFIRST	TO
SELECTGROUP	READREG
RESETFIRST	
RESETGROUP	
MATCHCOUNT	

5.2.4.3 Slave Instruction Implementation -- The following sections contain "pseudo-programs" which may be thought of as prototypes for the interpreter microprograms. In the interests of clarity, many details of information transfer have been omitted, as has the MSB/LSB-first issue in the arithmetic instructions.

5.2.4.3.1 Useful Routines --

- PROLOGUE (FIELD):

```
FIELD_DESCRIPTOR:=FIELD_TABLE_ORIGIN-/*Retrieve
                                         descriptor*/
FIELD_DES(FIELD);

IF FIELD_DESCRIPTOR.RT≠RECORD_TYPE /*Field in wrong
THEN ERROR;                        record type*/

FLENGTH:=FIELD_DESCRIPTOR.LENGTH; /*Capture length*/

FORDER:=FIELD_DESCRIPTOR.ORDER;    /*Capture order*/

RETURN;

END PROLOGUE;
```

- PUSH_STACK: /*Manipulating the match bit stack*/

```
IF MATCH.GT.STACKSPACE THEN OVERFLOW;

SAVEm(MATCH); /*Save previous value*/

MATCH:=MATCH+1;

RETURN;

END PUSH_STACK;
```

- POP_STACK: /*Manipulating the match bit stack*/

```
IF MATCH.EQ.0 THEN UNDERFLOW;

MATCH:=MATCH-1;

SELECTm (MATCH); /*Put new value in M*/

RETURN;

END POP_STACK;
```

- SAVE_MATCH_BITS: /*Copy match bits from word logic to record*/

SAVE_m (MATCH);

TO (MATCH_STACK_ORIGIN);

SELECT_m (15); /*Select words of correct format*/

SAVESTACK (MATCH)

 RETURN;

 END SAVE_MATCH_BITS;
- LOAD_MATCH_BITS: /*Copy match bits from record to word logic*/

TO (MATCH_STACK_ORIGIN);

SELECT_m (15); /*Selected word formats only*/

 ... /*Read occupancy bit and mask bit*/

LOADSTACK (MATCH)

SELECT_m (MATCH); /*Put top of match stack in M*/

 RETURN;

 END LOAD_MATCH_BITS;
- SELECT_WORDS: /*Find ECAM words of proper format*/

 IF FORMAT_CODE:=RECORD_DESCRIPTOR.

 /*Test FORMAT CODE from previous and bypass this when the new record instances are in the same words as the old record instances.*/

 FORMAT THEN RETURN;

TO (WORD_FORMAT_NUMBER);

 SET_m;

 MTOT;

 FORMAT_CODE:=RECORD_DESCRIPTOR.FORMAT;

B:=FIND_FREE_BUFFER;

COPY (FORMAT_CODE, B, 0); /*Copy string to beginning of
buffer*/

FOR M=1 TO LENGTH_OF_FORMAT CODE DO

/* We could build this into the iteration
box to set up match 15 and leave match
in the set bits*/

ATOMIC ARITH COMP (B);

END FOR;

MANDTTOM;

MTOMATCH (15);

CLEART; /*Save matched word indications in
match (15)*/

RETURN;

END SELECT_WORDS;

- ONE_STRING_PROLOGUE (STRING, FIELD); /*Common overhead
for operations with
one field operand*/

PROLOGUE (FIELD);

IF FLENGTH.NE. LENGTH(STRING) /*Length check*/

THEN ERROR; /*Justify or abort*/

B:=FIND_FREE_BUFFER; /*Get buffer for string*/

COPY (STRING, B, 0); /*Put string in buffer*/

TO (RECORD_DESCRIPTOR.ORIGIN /*Shift to field*/

+ FIELD_DESCRIPTOR.ORIGIN);

RETURN;

END ONE_STRING_PROLOGUE; /*Now all set up to per-
form function*/

- TWO_STRING_PROLOGUE (STRING1, STRING2, FIELD):
 - /*Two-field operand overhead*/
 - ONE_STRING_PROLOGUE (STRING1, FIELD);
 - IF 2*FLENGTH.GT.BUFFERLENGTH /*Strings won't fit in buffer*/
 - THEN ERROR;
 - IF FLENGTH.NE.LENGTH(STRING2)
 - THEN ERROR; /*Length mismatch error*/
 - COPY (STRING2, B, FLENGTH);
 - RETURN;
 - END TWO_STRING_PROLOGUE;
- LOOP: /*In prologue of every instr. that can be in automatic loop*/
 - IF LOOPING THEN RETURN; /*If in a loop now, do nothing*/
 - LOOPING:=TRUE;
 - RECORD_INSTANCE:=0; /*Initialize instance index*/
 - BIT_STACK_TOP:=MATCH; /*Save top of bit stack for looping*/
 - LOOP_LENGTH:=RECORD_TABLE_ORIGIN→/*Capture loop counter*/
 - RECORD_LIST_DESCRIPTOR(RECORD_TYPE).LENGTH;
 - LOOP_HEAD:=PC; /*Save origin of loop; comes back to this target instruction*/
 - RETURN;
 - END LOOP;
- DONT_LOOP: /*Used in prologue of every instr. that cannot be in automatic loop*/
 - IF LOOPING THEN RETURN;
 - SAVE_MATCH_BITS; /*Save last match bit stack*/
 - RECORD_INSTANCE:=RECORD_INSTANCE+1; /*Move to next instance*/

```

IF RECORD_INSTANCE.GE.LOOP_LENGTH THEN
    LOOPING:=FALSE          /*Terminate the loop*/
RETURN
RECORD_DESCRIPTOR:=RECORD_TABLE_ORIGIN-
    RECORD_LIST_DESCRIPTOR(RECORD_TYPE).ORIGIN-
    RECORD_DESCRIPTION(RECORD_INSTANCE);
                                /*Read descriptor of next instance*/
...          /*Check for same word format*/
SELECT_WORDS;          /*Mark match bit 15 in all words having
                        correct format*/
...          /*Read occupancy bit and use as mask*/
LOAD_MATCH_BITS; /*Copy match bit stack from ECAM to
                match bits in word logic*/
MATCH:=BIT_STACK_TOP;    /*Restore top of bit stack count*/
    PC:=LOOP_HEAD;        /*Cause looping on next fetch*/
END DONT_LOOP;

```

5.2.4.3.2 Instruction Implementations -- The following pseudo-programs are prototypes for operational microcode:

- FOR(FIELD): /*The options ALL, MATCHED, and LOCKED are tested within*/


```

DONT_LOOP;

IF LOOPING THEN RETURN;  /*Finish loop*/

PROCESS_TABLE_ORIGIN-PROCESS_TABLE_ENTRY
    (PROCESS_NUMBER).MATCH_STACK_DESCRIPTOR_TABLE_ENTRY
    (RECORD_TYPE):=MATCH//MATCH_STACK_ORIGIN;
                                /*Save status of match stack for the surrounding block*/

FIELD_DESCRIPTOR:=FIELD_TABLE_ORIGIN-
    FIELD_DES(FIELD);          /*Retrieve descriptor*/
      
```

```

RECORD_TYPE:=FIELD_DESCRIPTOR.RT; /*Set up record
                                     type*/
TOP_OF_STACK_PTR:=TOP_OF_STACK_PTR+1;
TOP_OF_STACK_PTR_WORD:=LOCKED//RECORD_TYPE;
                                     /*Stack new record type*/

MATCH_STACK_DESC:=PROCESS_TABLE_ORIGIN→
PROCESS_TABLE_ENTRY(PROCESS_NUMBER).
MATCH_STACK_DESC→MATCH_STACK_DESCRIPTOR_
TABLE_ENTRY(RECORD_TYPE); /*Retrieve match
                             stack information*/

STACKSPACE:=RECORD_TABLE_ORIGIN→ /*Length of space
                                   in record for each
                                   match stack*/

RECORD_LIST_DESCRIPTOR(RECORD_TYPE).

STACKSIZE;

IF MATCH_STACK_DESC=NULL

    THEN DO                                     /*Need a new stack*/

● SPACE_TEST: ALLOC_BITS:=RECORD_TABLE_ORIGIN→
    RECORD_TABLE_ENTRY(RECORD_TYPE).STACK_USAGE;

● IF ALLOC_BITS.NE.ALL_1s

    THEN DO                                     /*Stack space is available*/

        INDEX:=SCAN(ALLOC_BITS,'0'); /*Find first unused
                                         space*/
        RECORD_TABLE_ORIGIN→
        RECORD_TABLE_ENTRY(RECORD_TYPE).
        STACK_USAGE(INDEX):=1 /*Set allocation bit*/
        MATCH_STACK_DESC.ORIGIN:= /*Compute stack origin*/
        STACKSPACE*INDEX+STACK_SPACE_ORIGIN;
        MATCH_STACK_DESC.TOP:=0; /*Set top of new stack*/
    END_THEN; /*Note: Match stack descriptor will be
               completely saved at end of loop*/

```



```

ELSE DO
    SIGNAL_MASTER_TO_WAIT_FOR_STACK
        /*Send message to master that you cannot proceed until there is stack space
        available in this record type*/
    (RECORD_TYPE);
    WAIT;
    GOTO SPACE_TEST;      /*Try again on reactivation*/
END_ELSE;

END_DO;      /*End of stack allocation sequence*/

MATCH:=MATCH_STACK_DESC.TOP;  /*Capture top of
                                stack*/
MATCH_STACK_ORIGIN:=MATCH_STACK_DESC.ORIGIN;
                                /*Capture stack origin*/

• LOCK_TEST:    /*Note that the lock must be checked even if this
                  process won't lock it*/

IF RECORD_TABLE_ORIGIN=RECORD_TABLE_ENTRY
    (RECORD_TYPE).LOCK.NE.0^
    RECORD_TABLE_ORIGIN=RECORD_TABLE_ENTRY
        (RECORD_TYPE).LOCK.NE.PROCESS_NUMBER
    THEN DO      /*Already locked*/
        SIGNAL_MASTER_TO_WAIT_FOR_LOCK /*Await
        (RECORD_TYPE);                  unlocking*/
        WAIT;
        GOTO LOCK_TEST;
    END_DO;

```

IF LOCKED THEN

```
RECORD TABLE ORIGIN-RECORD      /*Set the lock. Note
    _TABLE_ENTRY(RECORD_TYPE)    that the separation
    . LOCK:=PROCESS_NUMBER;      works because this
                                sequence cannot be
                                interrupted, as no
                                process in the in-
                                struction interpreter
                                stops except by its
                                own volition -- it
                                being assumed that
                                the query translator
                                will produce "sound"
                                code.*/
```

/*However, note that deadlocks are possible in this scheme.*/

/*Process number is used in the lock to permit nesting
WHILES using the same record type; they would hang
up without a process number check.*/

IF ALL THEN /*Check the ALL vs MATCHED option*/

PUSH1; /*Execute the implied PUSH1*/

RETURN;

END_FOR; /*Note that the name FOR is ambiguous, being a
procedure -- defined here -- and an element of the
defining language. The context should make the
meaning clear.*/

- END_FOR: /*This is a procedure entry -- see note at end of
FOR definition.*/

DONT_LOOP;

IF LOOPING THEN RETURN; /*Finish loop*/

MATCH_STACK_DESC.TOP:=MATCH; /*Save top of match
bit stack*/

PROCESS_TABLE_ORIGIN-

PROCESS_TABLE_ENTRY(PROCESS_NUMBER).

MATCH_STACK_DES-MATCH_STACK_DESCRIPTOR

TABLE_ENTRY(RECORD_TYPE):=MATCH_STACK_DESC;

/*Save match stack descriptor*/

```

IF TOP_OF_STACK_PTR-WORD.LOCKED=1B
    THEN                                     /*Clear the lock*/
        RECORD_TABLE_ORIGIN←
            RECORD_TABLE_ENTRY(RECORD_TYPE).
            LOCK:=0;
            /*Now all state has been saved and we can restore
            the state of the surrounding block*/
        TOP_OF_STACK_PTR:=TOP_OF_STACK_PTR - 1; /*Pop the
                                                    stack*/
        RECORD_TYPE:=TOP_OF_STACK_PTR← /*Retrieve record
            WORD.SAVED_REC_TYPE;         type*/
        MATCH_STACK_DESC:=PROCESS_TABLE_ORIGIN←
            /*Retrieve match stack information*/
            PROCESS_TABLE_ENTRY(PROCESS_NUMBER).
            MATCH_STACK_DESC←MATCH_STACK_DESCRIPTOR_
            TABLE_ENTRY(RECORD_TYPE);
        MATCH:=MATCH_STACK_DESC.TOP;
        MATCH_STACK_ORIGIN:=MATCH_STACK_DESC.ORIGIN;
        RETURN;
    END END_FOR;

```

● FIND (STRING, FIELD):

```

    LOOP;
    ONE_STRING_PROLOGUE(STRING, FIELD); /*Housekeeping*/
    IF MATCH.GT.12 THEN ERROR; /*Any space for results?*/
    mTOT; /*Set up initial match bits*/
    ACOMPMSB(FLENGTH, B) /*Do compare*/
    mANDTTOm; /*Combine results*/
    RETURN;
    END FIND;

```

- GREATER (STRING, FIELD):
 - LOOP;
 - ONE_STRING_PROLOGUE(STRING, FIELD); /*See FIND for comments*/
 - IF MATCH.GT.13 THEN ERROR;
 - mTOT; /*Initialize match bit*/
 - ACOMPMSB (FLENGTH, B)
 - COMPLEMENTT;
 - mANDTTOT;
 - RETURN;
 - END GREATER;
- LESS (STRING, FIELD):
 - LOOP;
 - ONE_STRING_PROLOGUE(STRING, FIELD); /*See FIND for comments*/
 - IF MATCH.GT.13 THEN ERROR;
 - mTOT; /*Initialize match bit*/
 - ACOMPMSB (FLENGTH, B)
 - COMPLEMENTm;
 - mANDNOTTTOm;
 - RETURN;
 - END LESS;
- ADD (VALUE, FIELD):
 - LOOP;
 - ONE_STRING_PROLOGUE(STRING, FIELD);
 - PUSH_STACK; /*Save match bit*/


```

    ADDLSB (FLENGTH, B);
    TTOM;                                /*Overflow left in M*/
    RETURN;
    END ADD;
● SUBTRACT_FIELD (VALUE, FIELD):
    LOOP;
    ONE_STRING_PROLOGUE (STRING, FIELD):
    PUSH_STACK;                          /*Save match bit*/
    REVSUBLSB (FLENGTH B)
    TTOM;                                /*Put overflow in M*/
    RETURN;
    END SUBTRACT_FIELD;
● PUSH1:
    LOOP;
    PUSH_STACK;                          /*Get new entry*/
    SELECTm (15);                        /*Match (15) contains word selection bit*/
    RETURN;
    END PUSH1;
● PUSH0:
    LOOP;
    PUSH_STACK;
    CLEARm;
    RETURN;
    END PUSH0;

```

- POP:
 - LOOP;
 - POP_STACK;
 - RETURN;
 - END POP;
- DUPLICATE:
 - LOOP;
 - PUSH_STACK; /*This leaves previous value in M*/
 - RETURN;
 - END;
- EXCHANGE:
 - LOOP;
 - mTOT; /*Top*/
 - SELECTm (MATCH-1);
 - MEXCHANGET;
 - SAVEm (MATCH-1);
 - TTOM;
 - RETURN;
 - END;
- ROTATE3PLACES:
 - LOOP;
 - IF MATCH. LT. 2 THEN UNDERFLOW;
 - mTOT; /*T = M₁*/
 - SELECTm (MATCH-1); /*M = M₂, T = M₁*/

```

SAVEm (MATCH);          /*M = M2, T = M1; M1 rewritten*/
SELECTm (MATCH-2);       /*M = M3, T = M1; M1 rewritten*/
SAVEm (MATCH-1);         /*M = M3, T = M1; M1, M2 rewritten*/
TTOM;                   /*M = T = M1; M1, M2 rewritten*/
SAVEm (MATCH-2);         /*M = T = M1; all rewritten*/
SELECTm (MATCH);         /*Restore correct state in M*/
RETURN;
END ROTATE3PLACES;

```

- NOT:

```

LOOP;
COMPLEMENTm;
RETURN;
END NOT;

```

- OR:

```

LOOP;
mTOT;
POP_STACK;
mORTTOM;
RETURN;
END OR;

```

- AND:

```

LOOP;
mTOT;
POP_STACK;

```

mANDTTom;

RETURN;

END AND;

- XOR:

LOOP;

mTOT;

POP_STACK;

mXORTTOM;

RETURN;

END XOR;

Note: The foregoing list is not complete; the following instructions are among those that have not been implemented due to time constraints on the current effort:

- DECREMENT
- FASTREAD
- FASTWRITE
- READ
- WRITE
- GO_TO
- IF POSITIVE_GO_TO
- SAVE_BLOCK CALL
- ALLOCATE
- RETURN
- HALT
- WAIT
- ADD
- SUBTRACT
- COPY
- STRING COPY

- ADDRESS_COPY
- MAKE_POINTER
- SELECT_FIRST
- READ_FIRST
- WRITE_FIRST
- RESET_FIRST
- READ_FIRST_ELIMINATING_DUPLICATES
- MATCH_COUNT_TO
- STATUS_TO
- MAXIMUM
- MINIMUM

5.2.5 Firmware Register Assignments

This section specifies a possible assignment of emulation registers to the register modules provided in the interpreter (see Section 4.1). The important information is the module assignment, although specific assignments within the modules have also been given. To develop the assignments, the registers most frequently used are described; this leads to a tentative distribution of the registers into two independent single-port register files.

5.2.5.1 Information for Slave Registers -- The information that might reside in the slave's registers will be listed in groups based on the frequency of reloading the registers. Note that these frequencies can be quite different from the usage frequencies.

There are six different groups; the least frequently changed group will be listed first. Groups are designated within this list by the events causing their contents to (potentially) change:

- A. Modify data base structure or change descriptor tables
- B. Process switch
- C. Procedure call
- D. Enter FOR block
- E. Enter implicit loop
- F. Begin macroinstruction execution

Group A: This group includes registers holding the origins of system tables. Normally, the contents of these registers are not changed. The registers in this group are:

A1: Process Table Origin

A2: Record Table Origin

A3: Field Table Origin

All of these pointers are absolute addresses within the master's memory. If these tables should be length-checked, three additional registers holding the lengths should be assigned within the group. Each length register should be located in the same register file as the corresponding origin register.

Group B: This group includes registers holding information about the process being executed. The contents of these registers are changed when the master switches the controller to a new process. The registers in this group are:

B1: Process Number

B2: Stack Limit

B3: Stack Base

The process number is an integer used to index within certain tables. Stack location information is kept as absolute addresses within the master's memory. It is used to check for overflow and underflow.

Group C: This group includes registers holding information about the activation block corresponding to the current or next procedure invocation. The contents of these registers are changed during a procedure call code sequence and during the return operation. The registers in this group are:

C1: Base of Current Activation Block (AB)

C2: Next Value for C1

These quantities are absolute addresses within the master's memory. The contents of C2 are changed by the SAVE-BLOCK operator, which initiates calling sequences. The ENTER and RETURN operators modify C1, which is used primarily to locate local variables allocated stack space.

Group D: Registers in this group hold information about FOR blocks, which specify array records to be manipulated. The contents of these registers are changed by the FOR and END-FOR operators which bracket the blocks. The registers in this group are:

D1: Record Type Number

D2: Record Instance List Origin

D3: Record Instance Limit

D4: Match Stack Origin

D5: Match Stack Limit

The record type number and match stack limit are integers. The record instance list origin and limit are absolute master memory addresses. The match stack origin is a bit position index within array words.

Group E: Registers in this group hold information used to control implicit loops. The contents of these registers are changed upon entering the implicit loop or upon returning to the beginning of an implicit loop. All array operations that may be within implicit loops are considered to be within such loops (even though this may require that the loop contain only one operator). The registers in this group are:

E1: Saved Top of Match Stack Index

E2: Record Instance Descriptor Pointer

E3: Record Instance Origin

E4: Word Format Code

The index is an integer; it is used to reset the top of match stack index (F3) when the implicit loop is re-executed. The record instance descriptor pointer is a master memory address used to find the descriptor of the record instance used within the current loop. From this descriptor, the origin of the record instance (a bit index within the array words) is retrieved and the format code (an arbitrary bit pattern) used to select the appropriate array words.

Group F: Registers in this group hold information used within single macro-instruction interpretation cycles. The contents of these registers may be changed at arbitrary points within these cycles. The registers in this group include:

F1: Program Counter

F2: Top of Activation Block Stack

F3: Top of Match Stack

F4: Field Number

F5: Field Bit Origin

F6: Field Length

F7: Macro Function Code

F8: Instruction Bits No. 1

F9: Instruction Bits No. 2

F10: Instruction Bits No. 3
 F11: Instruction Bits No. 4
 F12: Operand Address No. 1
 F13: Operand Address No. 2
 F14: Operand Address No. 3
 F15: Operand Value No. 1
 F16: Operand Value No. 2
 F17: Buffer Number

5.2.5.2 Register Assignments to Register Files -- From the preliminary implementations of the slave machine instructions, it was determined which pairs of registers participated in binary operations, with two registers as data. These registers were then assigned to two files, resulting in Figure 111.

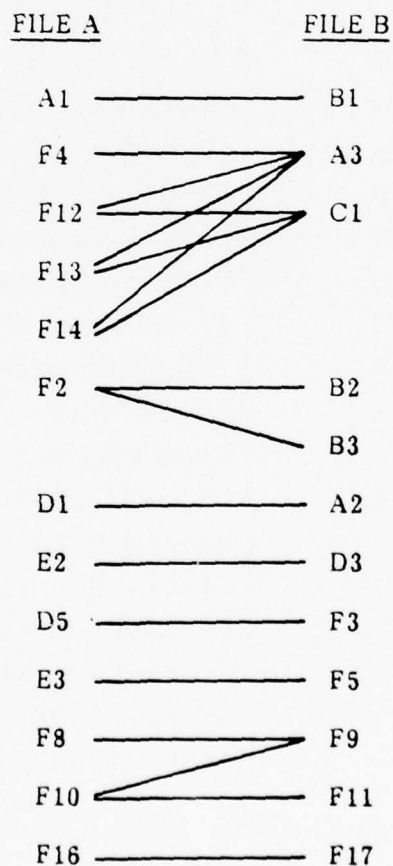


Figure 111. An Assignment of Registers into Two Files

showing an assignment of registers into two files (A and B) such that no binary operation requires two operands from the same file. Lines connect the pairs of registers that must be located in different files.

The final register assignment (Tables 50 and 51) is constructed by assigning those registers not appearing in Figure 111 to register files, using the constraints that: (1) those registers which supply binary operands used with constant (literal) operands should be located in File A (since literals are placed on the B bus); and (2) those registers supplying master memory addresses should be located in File A (since the master interface takes addresses from the A bus).

Table 50. Register Assignments within Register File A

Address (hex)	Code Name	Descriptive Name
20	A1	Process Table Origin
21	D1	Record Type Number
22	D2	Record Instance List Origin
23	D4	Match Stack Origin
24	D5	Match Stack Limit
25	E2	Record Instance Descriptor Pointer
26	E3	Record Instance Origin
27	E4	Word Format Code
28	F1	Program Counter
29	F2	Top of AB Stack
2A	F4	Field Number
2B	F6	Field Length
2C	F8	Instruction Bits No. 1
2D	F10	Instruction Bits No. 3
2E	F12	Operand Address No. 1
2F	F13	Operand Address No. 2
30	F14	Operand Address No. 3
31	F16	Operand Value No. 2
32-3F	---	Unassigned/Temporary Storage

Table 51. Register Assignments within Register File B

Address (hex)	Code Name	Descriptive Name
60	A2	Record Table Origin
61	A3	Field Table Origin
62	B1	Process Number
63	B2	Stack Limit
64	B3	Stack Base
65	C1	Base of Current AB
66	C2	Next C1 Value
67	D3	Record Instance Limit
68	E1	Saved Top of Match Stack Index
69	F3	Top of Match Stack
6A	F5	Field Bit Origin
6B	F7	Macro Function Code
6C	F9	Instruction Bits No. 2
6D	F11	Instruction Bits No. 4
6E	F15	Operand Value No. 1
6F	F17	Buffer Number
70-7F	---	Unassigned/Temporary Storage

SECTION 6

HARDWARE IMPLEMENTATION

The ECAM system will consist of three basic cabinet types:

- Memory Unit cabinet
- Control Unit cabinet
- Peripheral cabinet

A minimum system will have one each of these cabinets and a maximum Memory Unit storage capability of 10 million bytes. Expansion to 160 million bytes will be accommodated by adding additional Memory Unit cabinets up to a total of 16 per Control Unit. Each Memory Unit will be interfaced via differential twisted-pair cabling into dedicated buffer boards in the Control Unit.

A typical ECAM system is shown in Figure 112. All Control Unit and Peripheral cabinetry and other hardware is of the standard 19-inch-wide variety. The Memory Unit cabinet is necessarily slightly wider - 23 inches. Commercial-grade parts are used wherever possible to lower system cost. A floor plan concept for a 160-million-byte system (16 Memory Unit cabinets) is shown in Figure 113. Such a system would require approximately 400 square feet of floor space. Total power requirements would be approximately 100 kilowatts at 230 volts, 60 cycles.

6.1 MEMORY UNIT CABINET

The Memory Unit cabinet will contain up to 10 million bytes of storage capability. The functional organization will be 20,480 serial words of 4096 bits each. The physical partitioning will be 32 storage boards with 640 words per board. Each group of eight storage boards will be driven by one of four Storage Buffer boards which will in turn be driven by a single Memory Unit cabinet buffer board. These five boards, along with a small amount of logic on each storage board and all of the back panel wiring, cabling, and termination, will comprise the signal distribution system within the Memory Unit cabinet. A diagram of the physical interconnect scheme for the Memory Unit is shown in Figure 114. An artists conception of the Memory Unit is shown in Figure 115.

The total board complement of 38 boards is comprised as follows:

- 32 storage unit boards at 85 watts per board
- 4 storage buffer boards at 4 watts per board (Memory Unit Signal Distribution Board A)

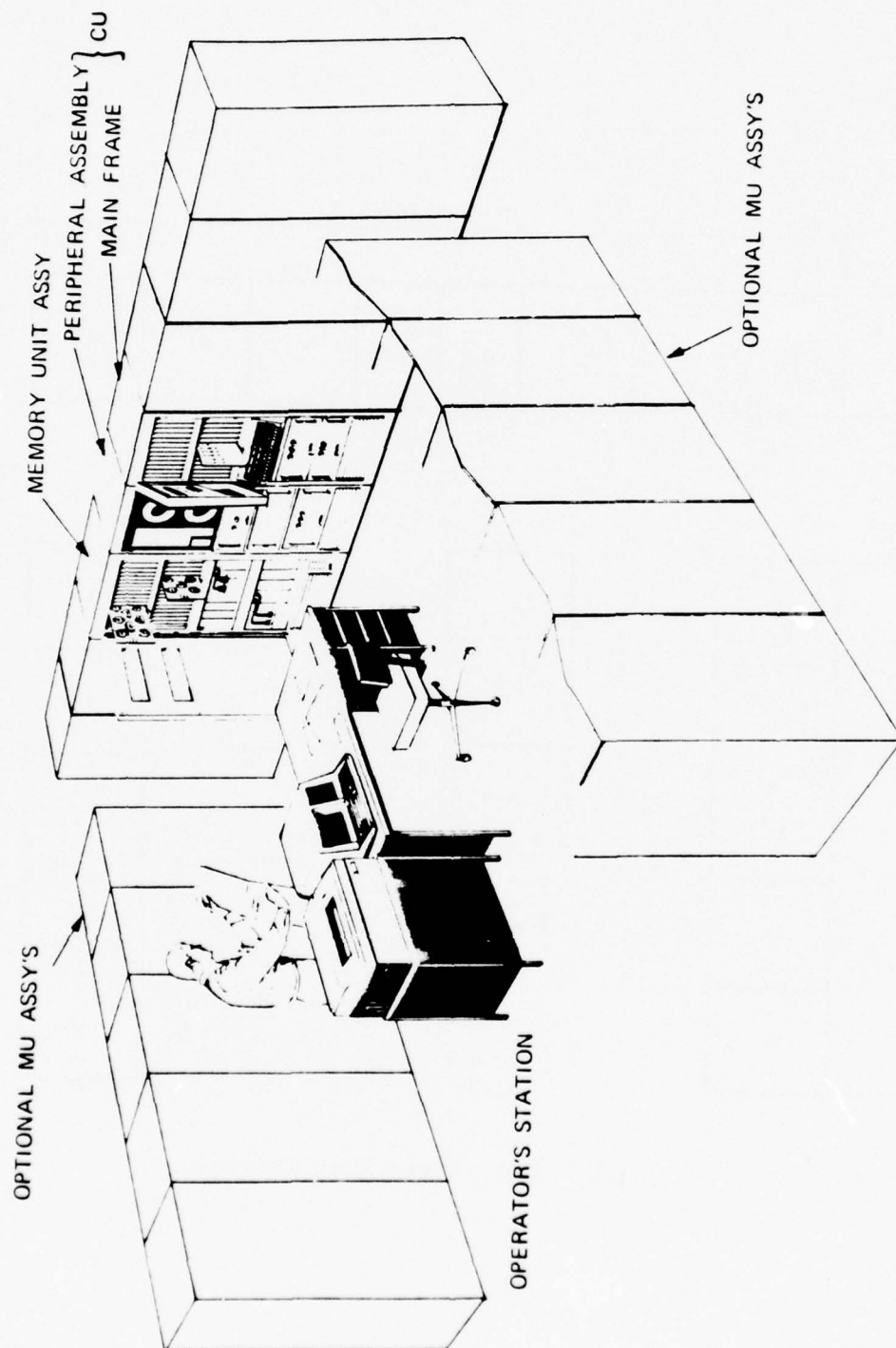


Figure 112. Typical ECAM System

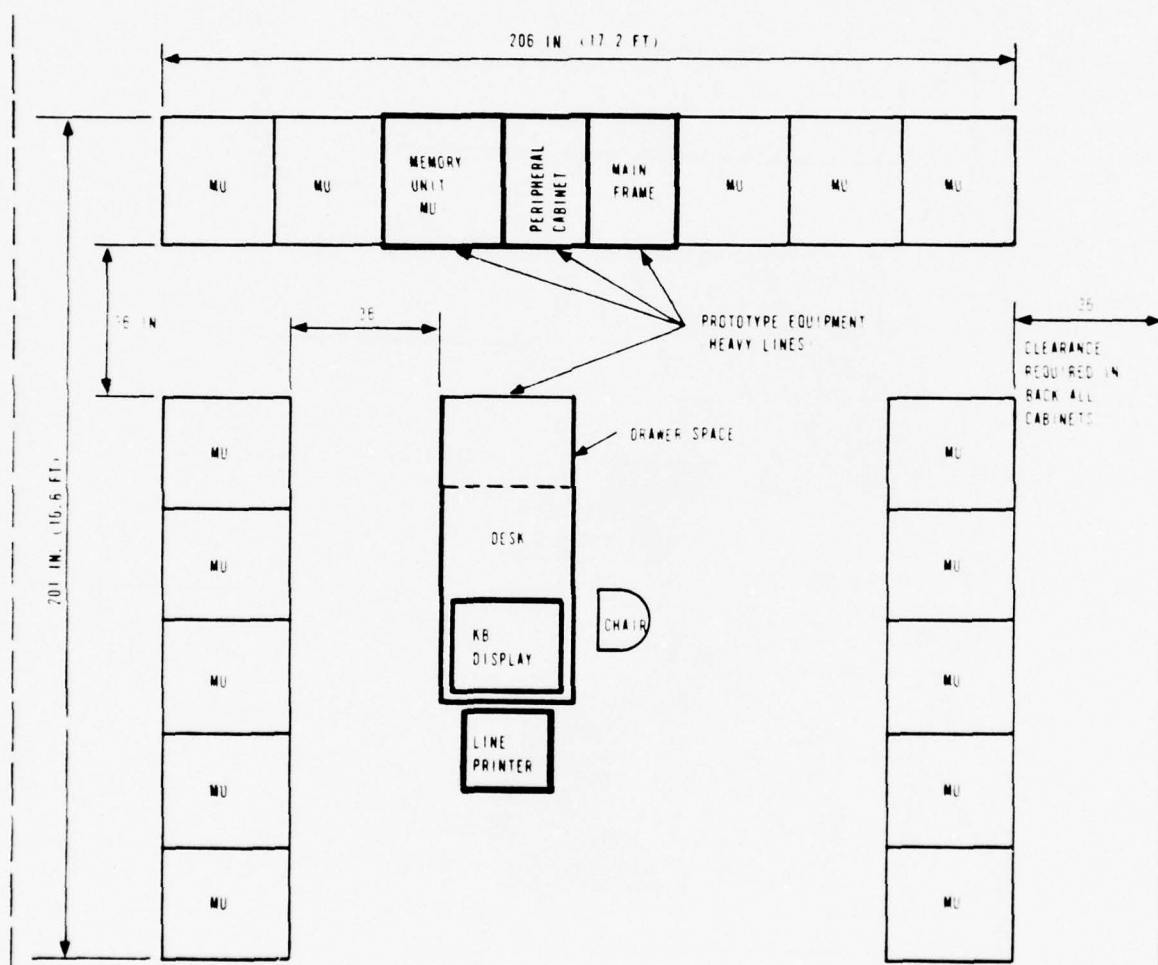


Figure 113. Floor Plan Concept

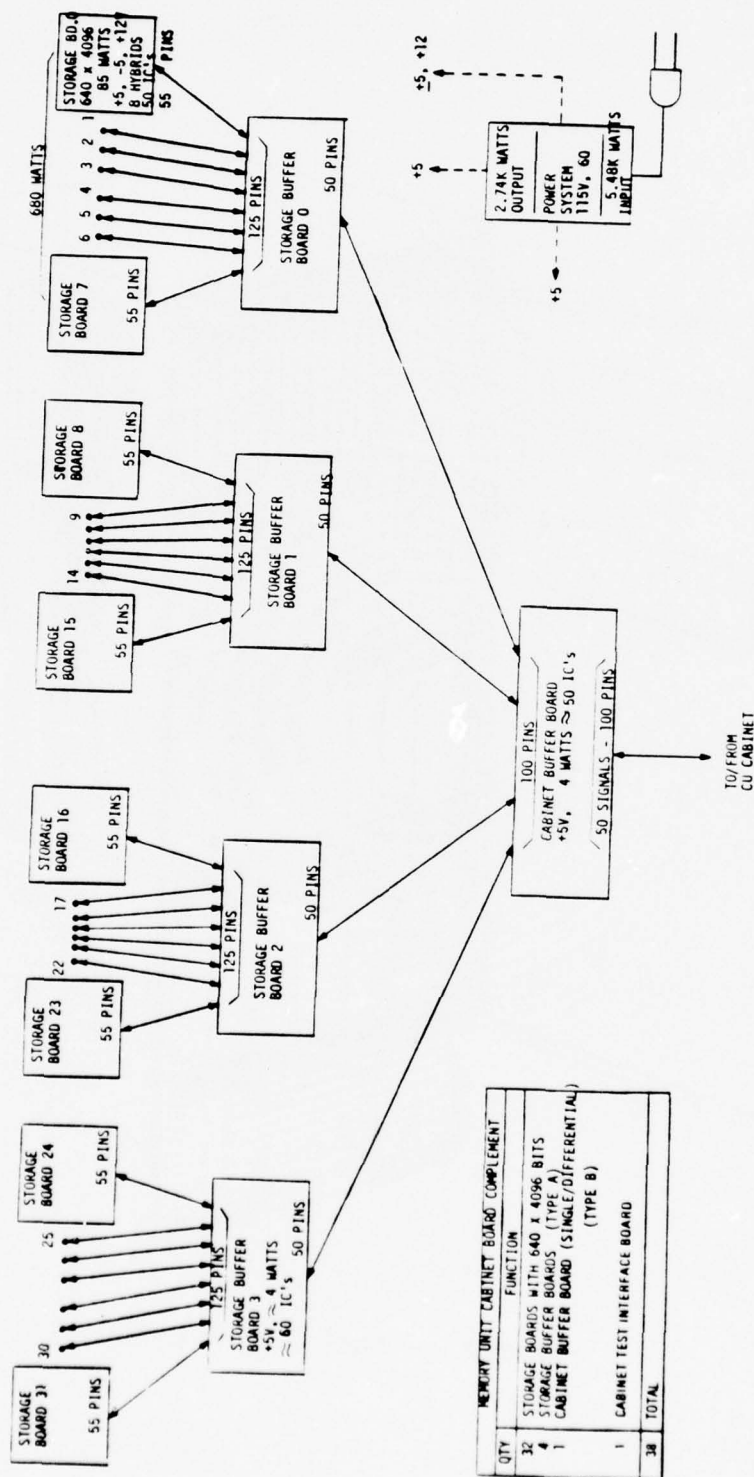


Figure 114. Memory Unit Cabinet Interconnect Scheme

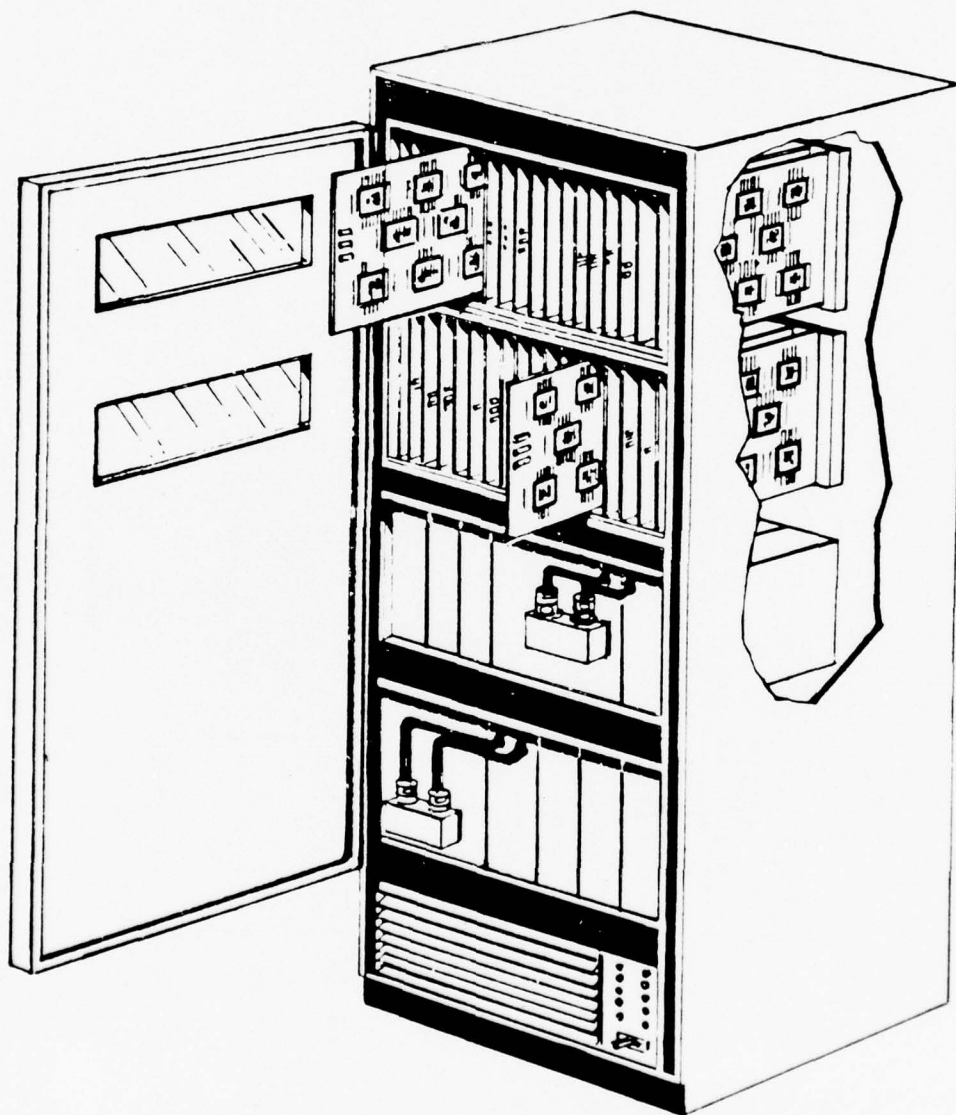


Figure 115. Memory Unit Cabinet

- 1 cabinet buffer board at 4 watts per board (Memory Unit Signal Distribution Board B)
- 1 cabinet test interface board at 5 watts per board

Each board will be 12 inches x 16 inches with 200 I/O connector pins each. All boards will be housed in two card cages with space for up to 19 boards per cage. Figure 116 shows the arrangement of cards in the Memory Unit cabinet.

6.1.1 Power Supplies and Cooling

Figure 117 is the Memory Unit layout showing location of boards, power supplies, and blowers. Required voltages of +5, -5, and +12 are provided by power supplies selected from the Honeywell 6000 Series computer main-frame system. These supplies are manufactured commercially at the Deer Valley Plant of the Honeywell Phoenix Computer Operations. Dual Rotron MB-8100 cabinet panel blowers that deliver air at the rate of 500 cubic feet per minute each were selected for the Memory Unit cabinet. Air ducting in the cabinet will be designed to provide room temperature inlet air to the power supplies and boards separately. Total air temperature rise through the cabinet will be approximately 20° F.

6.1.2 Storage Board

The ECAM architecture was initially conceived and developed around the Contractor-developed T-10(superchip) charge-coupled device (CCD). The T-10 has been developed to the point of prototyping the chip, but further development is questionable at this point. A major problem related to the T-10 was identified early in the ECAM development when the requirement for Read/Modify/Write (R/M/W) of the storage device became apparent. The T-10 does not have this capability, but could be modified to incorporate it. The initial baseline design of the storage board included eight 3-inch x 3-inch hybrid modules with eight T-10 CCD chips per hybrid. The T-10 is organized as 10 serial words of 4096 bits each. Each hybrid had eighty 4096-bit words and each board had 640 words. An example of this hybrid organization is shown in Figure 118.

Recognition of the R/M/W problem in the T-10 led to consideration of other storage alternatives which might already be commercially available and which might be easily accommodated without significantly changing the architecture. The ECAM architecture is quite independent of the storage media and therefore provides the capability to use serial storage devices per the state of the art at the time of actual system build.

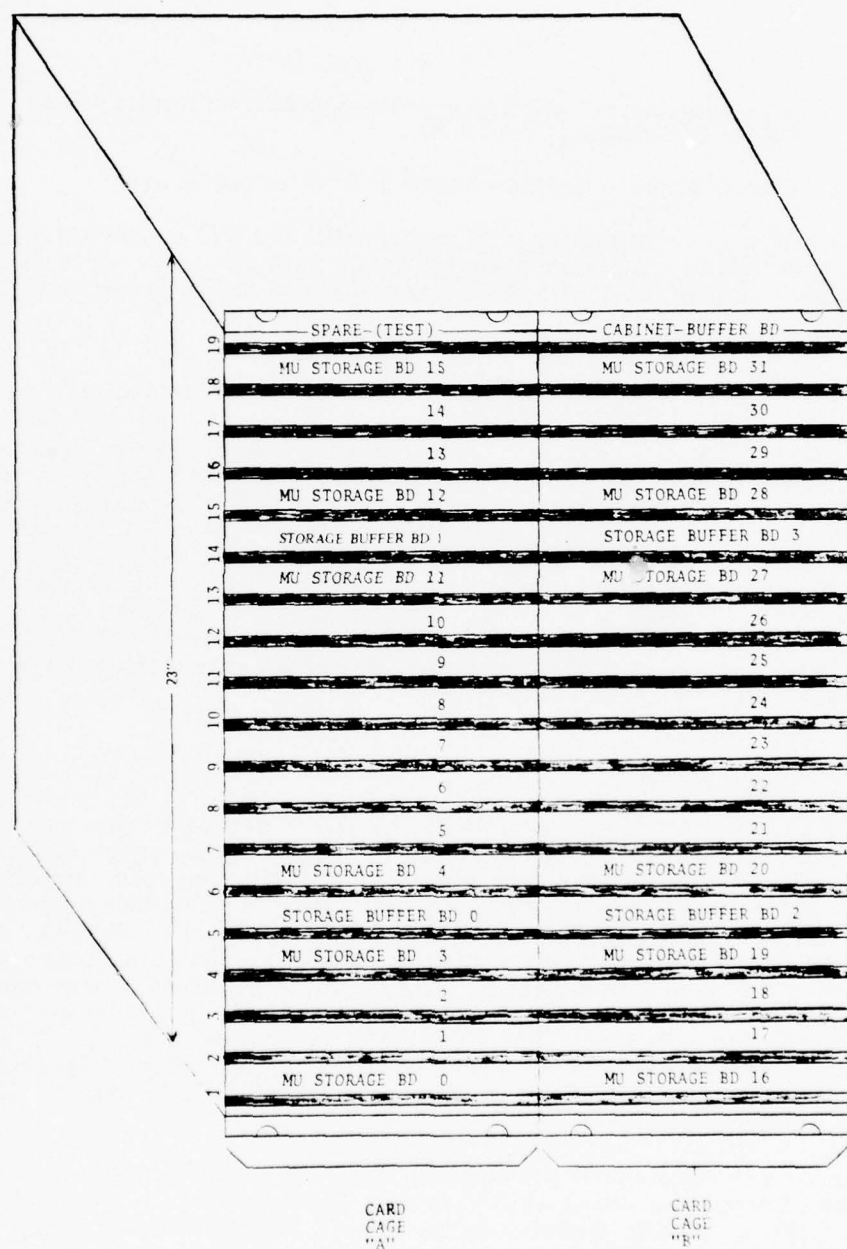


Figure 116. Memory Unit Card Assignments

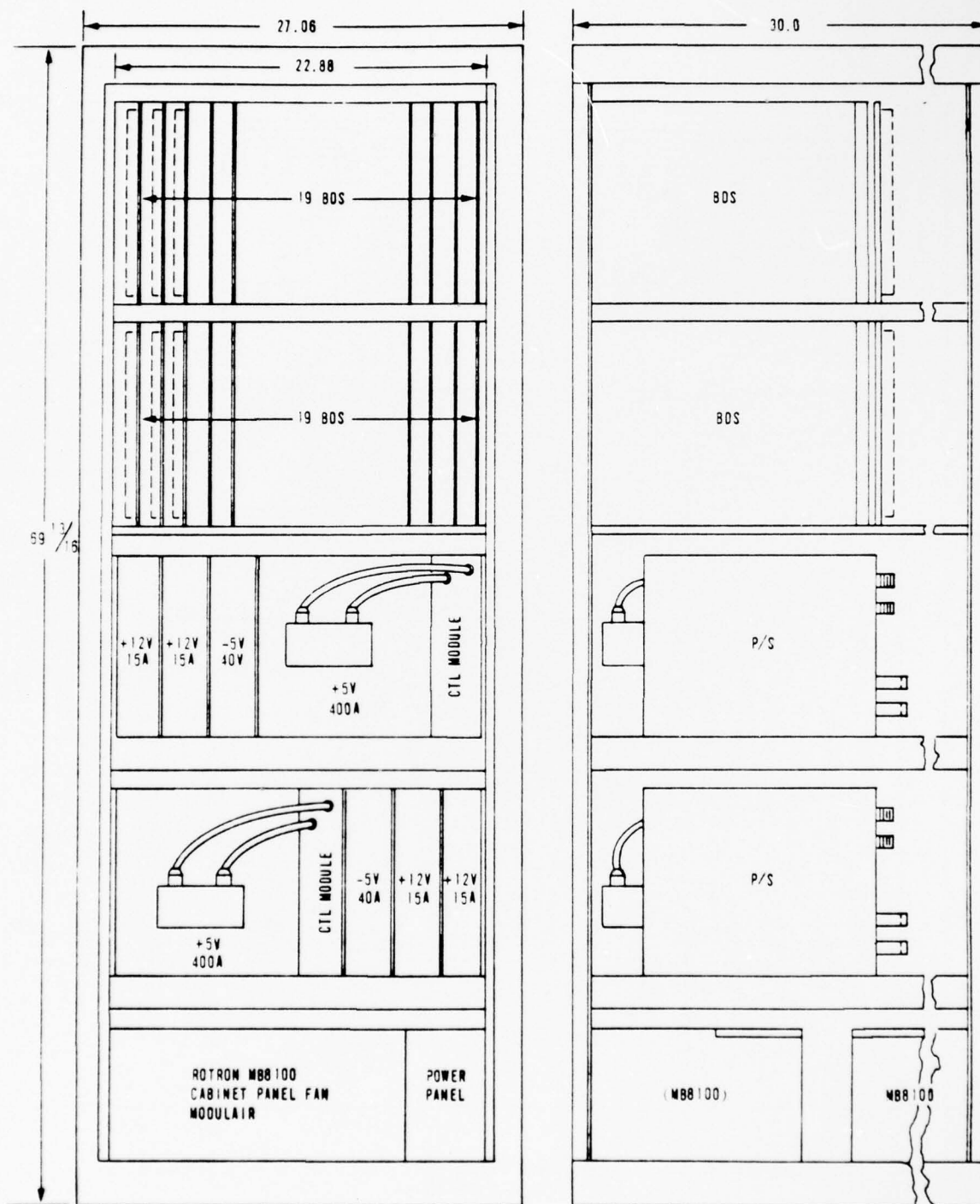


Figure 117. Memory Unit Cabinet Interior Layout

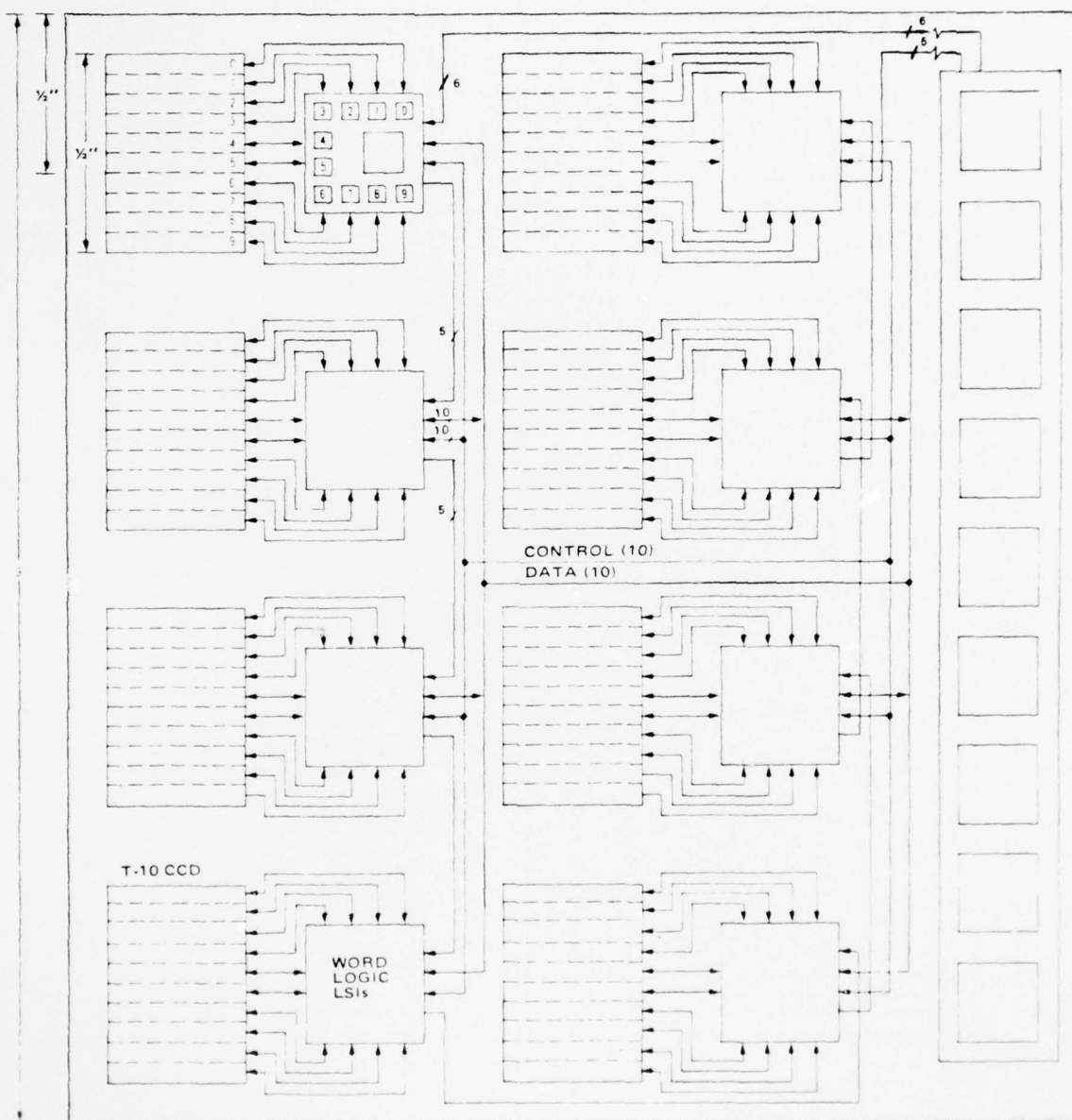


Figure 118. 3-inch x 3-inch Hybrid with T-10 Chip as Storage Device

Our current packaging baseline storage device is the Fairchild 4096 x 4-bit CCD (CD-460). This 16K-bit device is a dynamic N-channel device with a shift rate of up to 5 MHz. The CD-460 is transistor-transistor logic (TTL)-compatible at all I/O pins except for the two-phase clock pins. It has four operating modes:

- Read
- Write
- Read/Modify/Write
- Recirculate

and has a maximum power usage of 200 mW. The CD-460 has a chip size of 201 x 219 mils and comes in a standard dip package of 22 pins. The baseline packaging scheme for the storage devices in the ECAM system is to use hybrid technology on a 12-inch x 16-inch motherboard (Figure 119).

An alternate packaging approach would be to use dip packaging. This approach would double the board count or, conversely, halve the number of 4096 bit words per 12-inch x 16-inch board to 320 instead of 640.

For systems that do not require as much memory storage or that do not have a space problem, the dip packaging approach might very well be the most cost effective. Also, there are expectations for a 65K version of the CD-460 to be available within 12 months (1977). This is a fourfold increase which will have the same organization (four data I/O), but which will have 16K bits per word. This will quadruple the amount of storage capability in the same volume, but will also increase the ECAM search time by about the same factor due to the loss of parallelism. The speed versus volume versus cost tradeoff will have to be considered very carefully at the time of actual implementation, since it is somewhat application-dependent.

The hybrid module approach described below will use a 3-inch x 3-inch hybrid package with 80 words of 4096 bits each (327,680 bits). Each board will contain eight hybrids, with additional control and buffering logic in the form of discrete logic dips (approximately 50 dips). A functional block diagram of the Memory Unit storage board, using hybrids, is shown in Figure 120. The board will be approximately 12-inches x 16-inches, will have approximately 200 I/O pins (estimated 55 required), and will dissipate approximately 85 watts of power.

6.1.2.1 ECAM Hybrid -- Each storage board will contain eight 3-inch x 3-inch hybrids (Figure 121) which will be organized as 80 words of 4096 bits each. This organization still reflects the initial intentions of using the T-10 (10-channel) CCD. Not much is changed by the use of the CD-460 except that instead of a single 500-mil x 500-mil chip connected to every 10-channel WLLC, there will be 2.5 four-channel CCDs connected to every WLLC (Figure 122). Minor control signal changes are also required. Total parts complement for the 3-inch x 3-inch hybrid is:

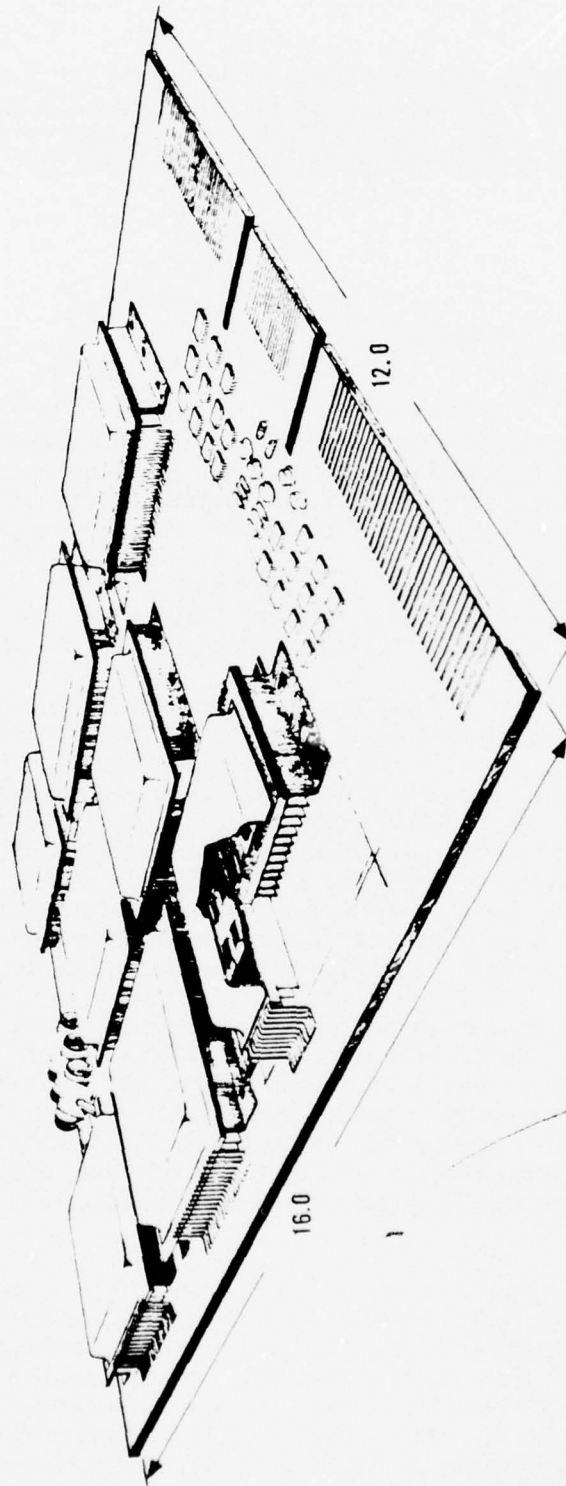
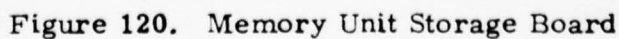


Figure 119. Memory Board Assembly



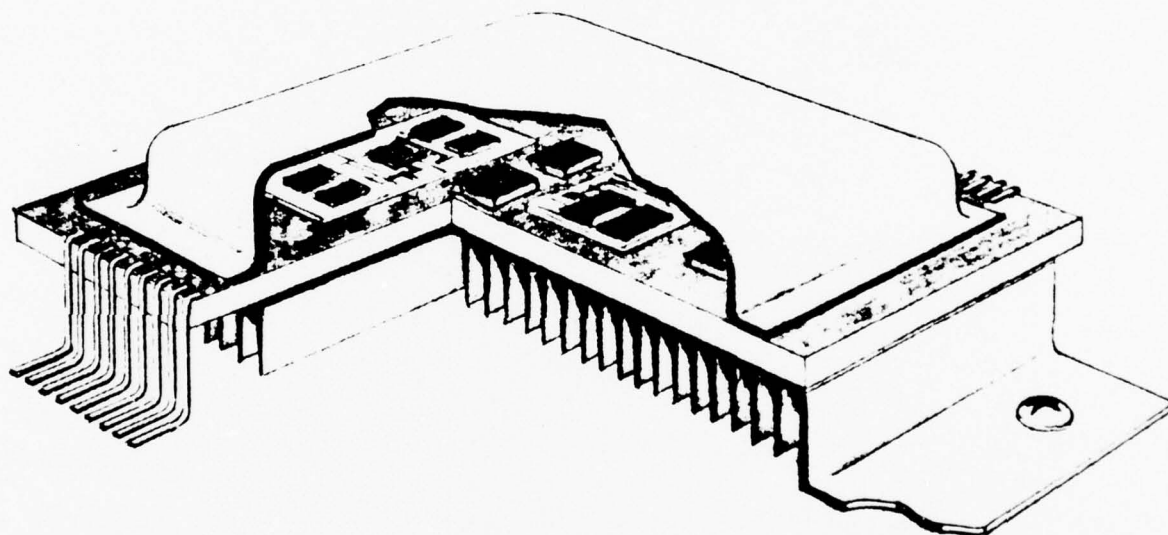


Figure 121. Memory Unit Hybrid Assembly

- 20 CD-460 CCD storage devices
- 8 custom word logic LSICs
- 8 clock driver chips
- Approximately 10 control and data buffer chips

A motherboard approach will be used in the hybrid package. Two subunit types will be developed: a storage subunit, and a buffer subunit. Each hybrid will have four storage subunits and one buffer subunit. Each storage subunit will contain 20 words (two WLLCs and five CCD chips). If a different CCD storage device (i. e., not CD-460) is used, a redesign of the storage subunit should accommodate the differences with no impact on the rest of the hybrid. More drastic changes, such as switching to RAM or Bubble devices, will probably require more significant redesign of the hybrid.

Total hybrid power dissipation is expected to be approximately 10 watts. All ICs are commercially available with the exception of the WLLC custom LSIC, which is the main control chip for ten 4096-bit words. Each hybrid will require approximately 42 pins for I/O, control, and power. A detailed dimension drawing of the hybrid package is shown in Figure 123.

6. 1. 2. 2 Word Logic LSIC (WLLC) -- The word logic LSIC contains the main control logic for 10 words of serial memory. A sample layout of the WLLC LSIC is shown in Figure 124. As indicated in the figure, each word has a certain amount of unique control logic associated with it. Also, the control and switching matrix for DIO is shown. Additional gating and buffering (drivers) for control signals will be necessary as well.

An N-channel silicon gate process was used for estimating the LSIC chip sizing. This process has been used at Honeywell's Contractor's Solid State Electronics Center and is known to be stable with consistent, predictable, results experienced over many process runs. For this estimate, enhancement-type clocked loads were assumed. For the final configuration, depletion loads will also be considered as a possible means to minimize clocking requirements and improve speed.

The estimated chip size of 213 mils square is broken down according to function in Table 52. To estimate interconnect, the active area is doubled. Input and output pads plus buffers and scribe line are included in a 20-mil-wide band around the periphery of the chip. Operating power estimate is 0.5 watt for a 1-MHz clocking rate.

6. 1. 2. 3 Storage Alternatives -- There are several possible alternatives for the storage device in an ECAM system. As mentioned, system requirements such as access time and word (record) size are important considerations. Also, CCDs have a potentially limiting shortcoming in the inability to shift in either direction. For certain systems, a need to shift bidirectionally could force the use of devices such as the 16K x 1-bit RAM. The additional requirement for nonvolatility could make the bubble memory device very attractive.

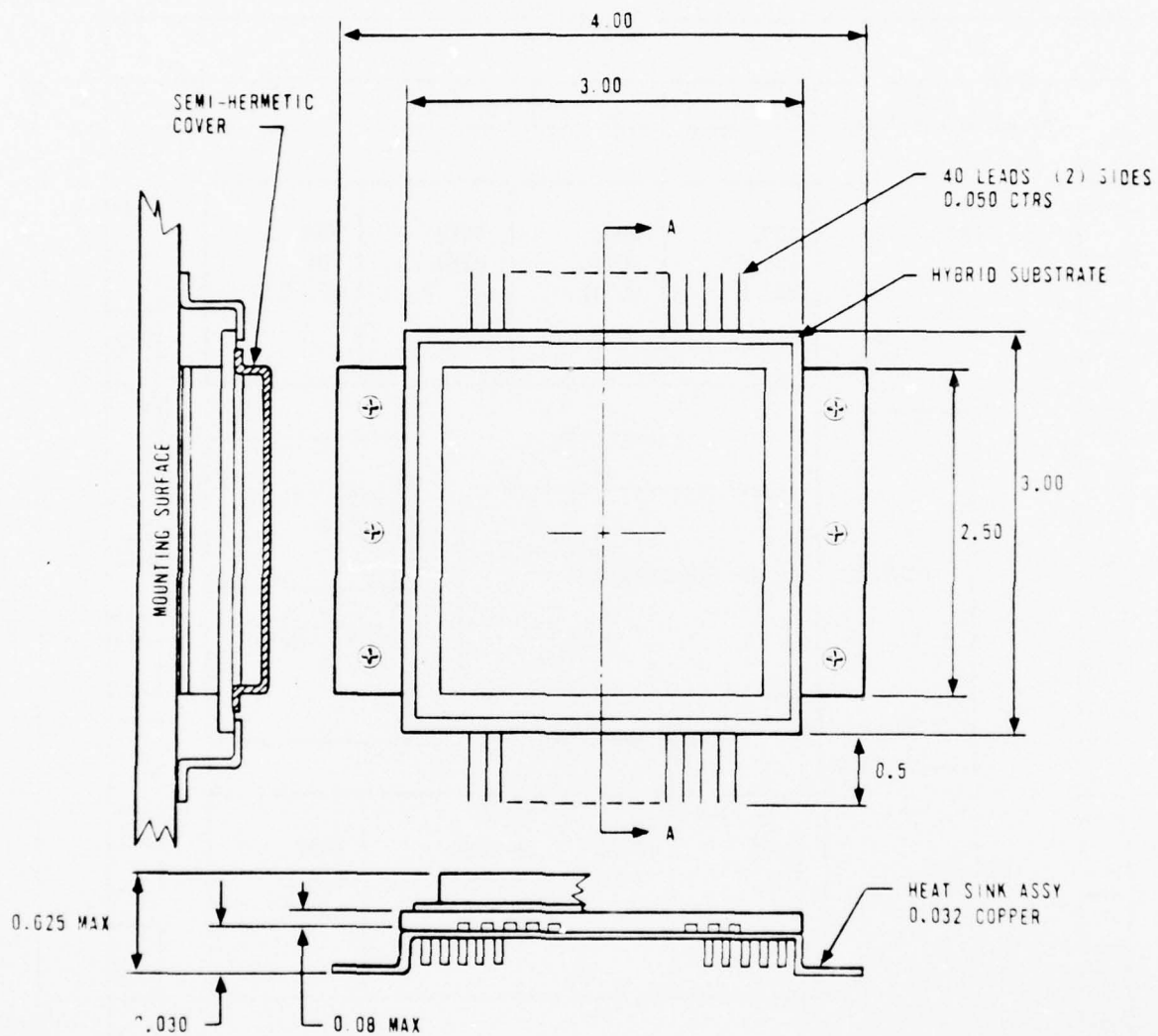


Figure 123. Hybrid Packaging Concept

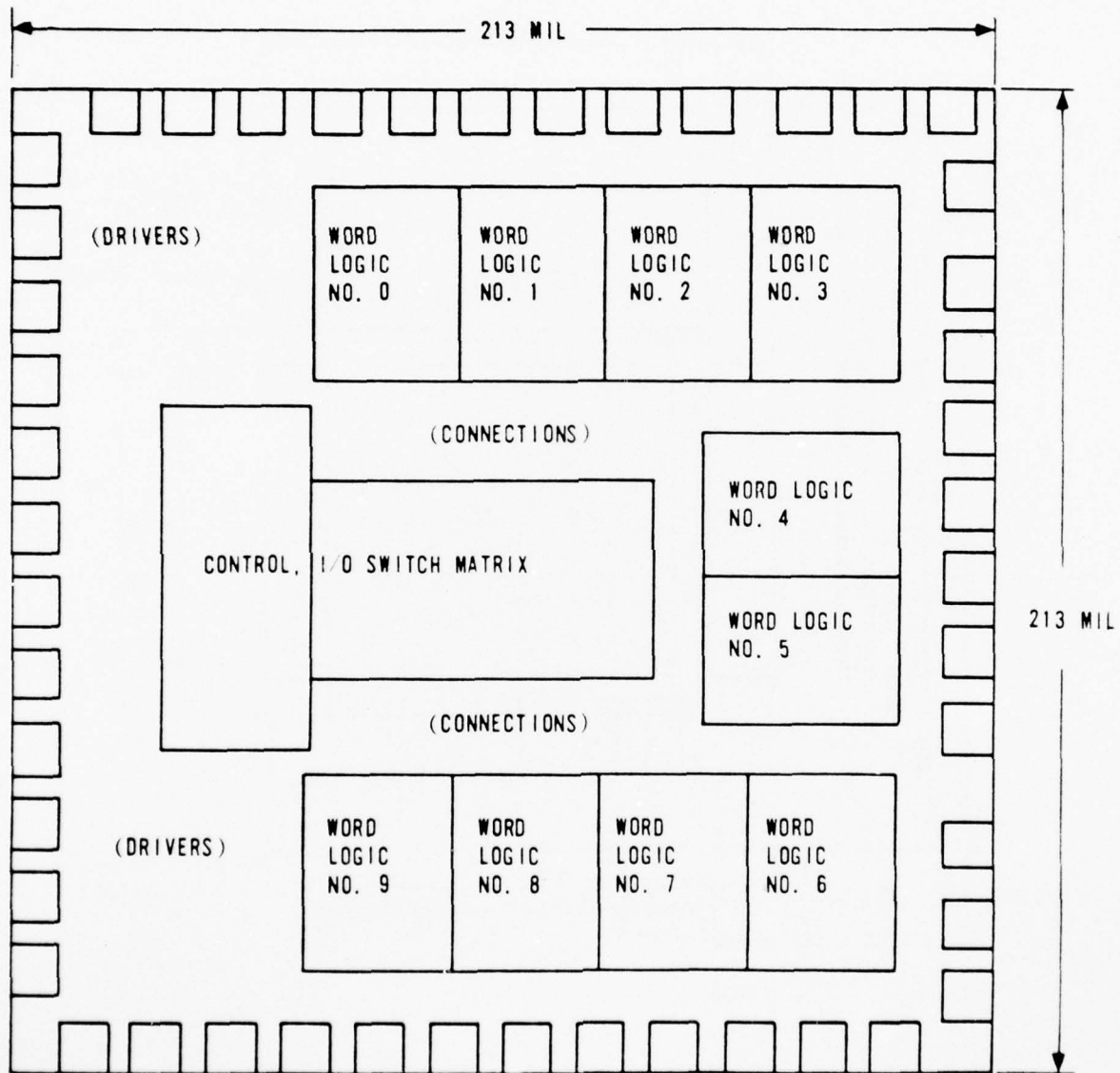


Figure 124. Word Logic LSIC

Table 52. ECAM Word Logic LSIC Estimated Area^a

Function	Size (mils)
• Word Logic Block, 1000 mil ² x 10	10,000
• I/O Switch Matrix, Encode, Decode, Selector, and Adder	1,960
• Match RAM, Selector, Registers, and Decode	2,980
Subtotal	14,940
• Double for Interconnects	x 2
Total	29,880

^a Chip size = 173 mils square. Include pads and buffers (20 mils/side). Total chip size = 213 mils square.

The basic ECAM architecture can accommodate any of these storage devices with minor changes in the control unit, signal distribution, and in the word logic control function. Each of the three types of storage elements considered for implementation in ECAM (i. e., CCDs, magnetic bubbles, and semiconductor RAMs) has certain unique features to offer the system. These features are summarized in Table 53.

Table 53. Storage Device Alternatives

Parameter	Characteristic		
	CCD	Bubble	RAM
Addressing mode	Serial	Serial	Random
Data retention	Dynamic/volatile	Static/non-volatile	Dynamic/volatile
Shift capability	Unidirectional	Bidirectional	Bidirectional
Transfer rate	2 to 10 MHz	100 kHz	2 to 4 MHz
Chip capacity (1976)	64K bits	256K bits	16K bits
Power required for 1 x 10 ⁶ bits	16 to 20 watts	9 watts	45 to 50 watts
Chip size	201 x 219 mils	(See Note)	145 to 235 mils

Note: The bubble device is a pilot production device manufactured by TI in a dip package containing sixteen 16K-bit bubble device chips.

This range of access time and size/weight/power/density considerations is certain to expand even more as the CCD and bubble technologies continue to mature.

6.1.3 Storage Buffer Board (Memory Unit Signal Distribution Band A)

The second board type in the Memory Unit cabinet is the storage buffer board (SBB) (Figure 125). The SBB serves as a control signal buffer for eight storage boards and contains the fourth, fifth, and sixth levels of the CGL tree which does memory match resolution and responder counting.

Altogether, the SBB requires approximately 175 I/O pins and has approximately 50 TTL ICs. Data I/O is single-ended tri-state, both upstream to the storage boards and downstream to the cabinet buffer board (CBB). All other signals are TTL-compatible. The board will dissipate approximately 4 watts of 5-volt power.

6.1.4 Cabinet Buffer Board (Memory Unit Signal Distribution Board B)

The CBB is the third board type in the Memory Unit cabinet. It has upstream I/O and control connections to the four SBBs and connects downstream to a counterpart buffer board in the Control Unit. The CBB converts differential signals going to and from the Memory Unit cabinet to single-ended signals within the cabinet. It also contains the seventh and eighth levels of the CGL tree.

The CBB will require approximately 200 I/O pins, will use 45 to 50 ICs, and will dissipate approximately 4 watts of 5-volt power. A functional block diagram of the CBB is shown in Figure 126.

6.1.5 Test Interface Board

The test interface board is currently defined only to the extent that it exists as a spare board slot and could contain simple exercisor logic or possibly the interface logic to a more sophisticated tester. Issues of ECAM system test and diagnostics are covered in Section 6.4.

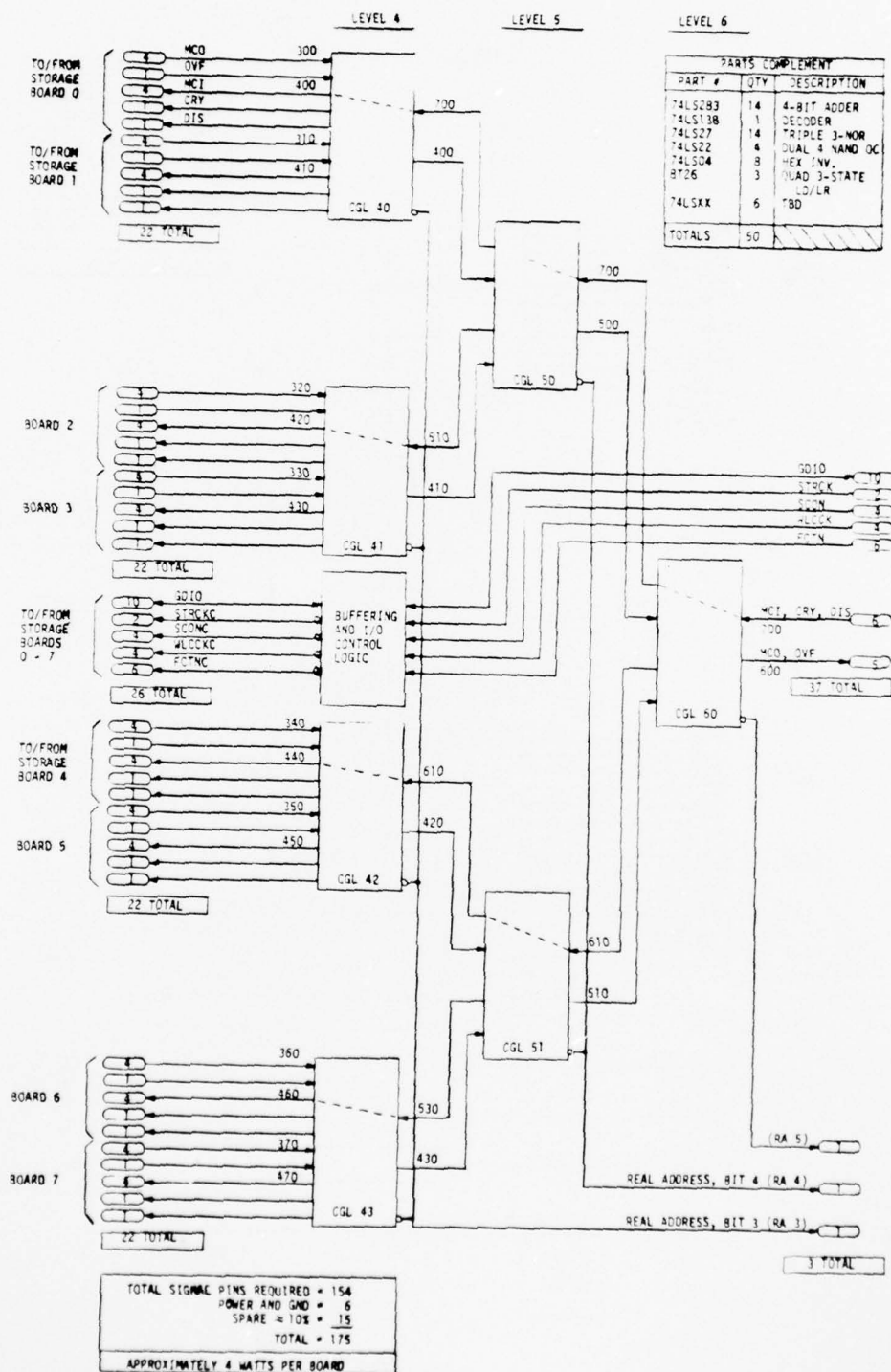


Figure 125. ECAM Memory Unit Storage Buffer Board (Signal Distribution Board A)

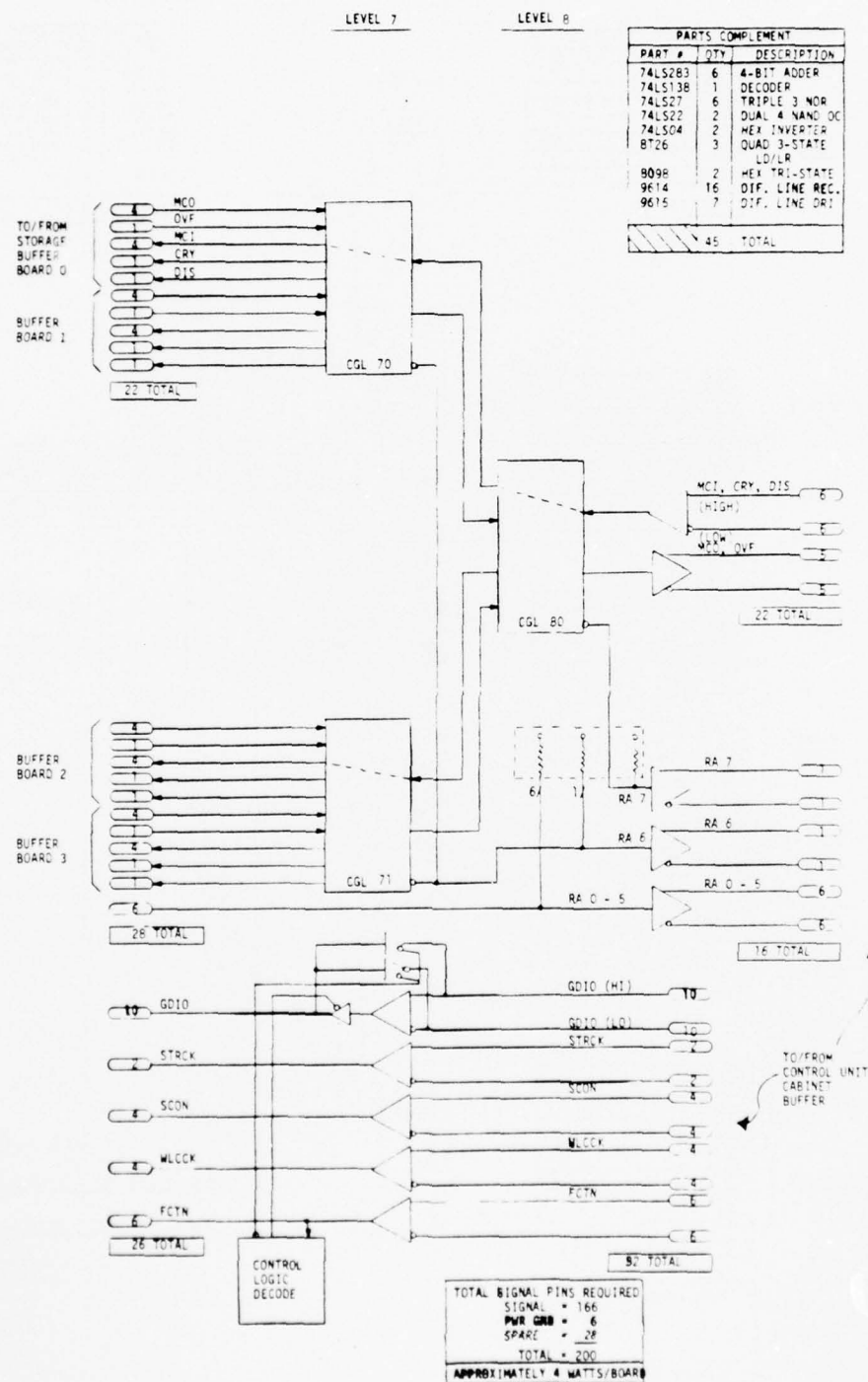


Figure 126. Memory Unit Cabinet Buffer Board
(Signal Distribution Board B)

6.2 CONTROL UNIT CABINET

The Control Unit cabinet will contain the following functional units:

- Host (6080) interface subunit, including the DMA port for fast I/O.
- Master control processor - currently baselined as a PDP-11/45, but the newly developed Honeywell Level 6 minicomputer system is also a candidate.
- Slave control processing unit:
 - Slave control processor
 - Storage controller
 - Fast I/O controller
 - Data/instruction buffers
 - Iteration controller
- Cabinet buffer logic - containing CGL network and line drivers/line receivers.
- Control panel interface logic paddle board.

An artist's conceptual drawing of the Control Unit cabinet is shown in Figure 127; a detailed dimension drawing is shown in Figure 128. The Control Unit cabinet will be basically the same design as the Memory Unit cabinet from a mechanical packaging and cooling point of view. Power supplies and power distribution will be the same as in the Memory Unit cabinet also. The card cages and card sizes will be the same (e. g., 12-inch x 16-inch circuit cards) with the exception that the cards will be of the dip socket/wire-wrap interconnect variety as opposed to the more specialized printed-circuit board which will be used for the Memory Unit storage board. Also, the card cages will be 19-inches wide in the Control Unit as opposed to 23 inches in the Memory Unit.

Board assignment for the Control Unit card cages is shown in Figure 129. Maximum board complement will be twenty-six 12-inch x 16-inch boards as follows:

- | | |
|---|---|
| ● Control Unit cabinet signal distribution boards | 8 |
| ● Master signal distribution board | 1 |
| ● Host interface | 2 |
| ● Slave control processor | 2 |
| ● Storage controller | 1 |

AD-A037 834

HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 9/2
ADVANCED LOGIC TECHNOLOGY.(U)

UNCLASSIFIED

FEB 77 'G A ANDERSON, E D JENSEN, R Y KAIN

F30602-75-C-0148

F0375-FR

RADC-TR-76-388

NL

4 of 4
ADA037834



END

DATE
FILMED

4-77

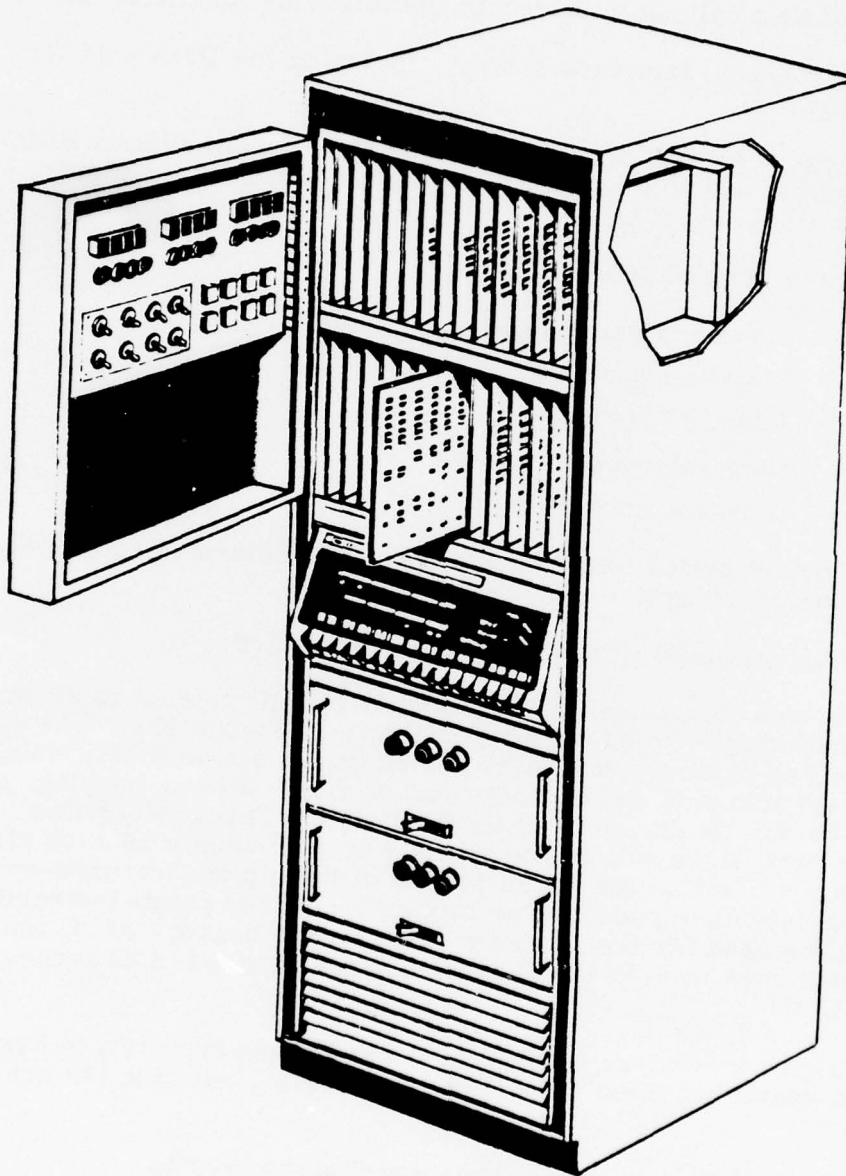


Figure 127. Control Unit Cabinet

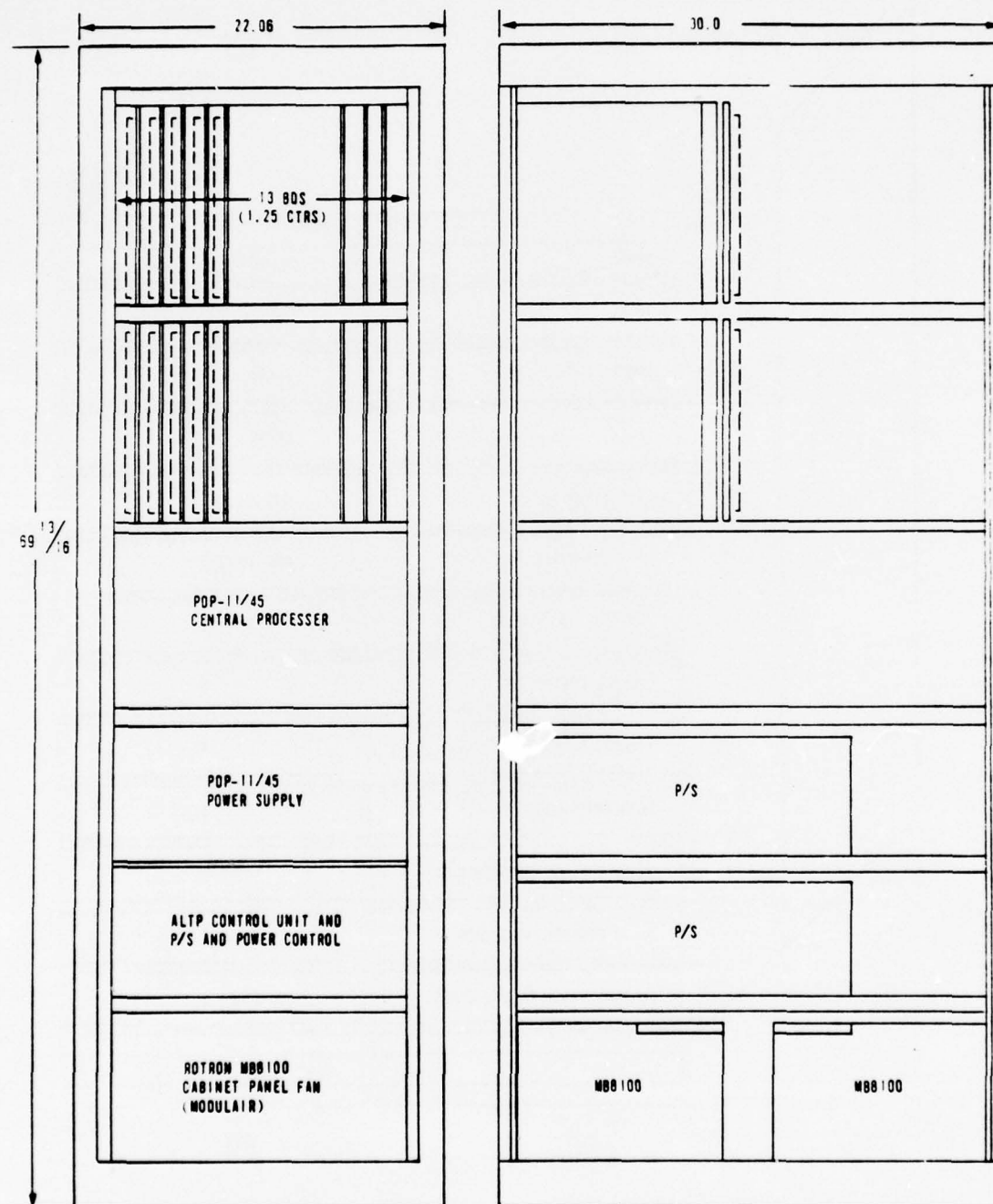


Figure 128. Control Unit Cabinet Interior Layout

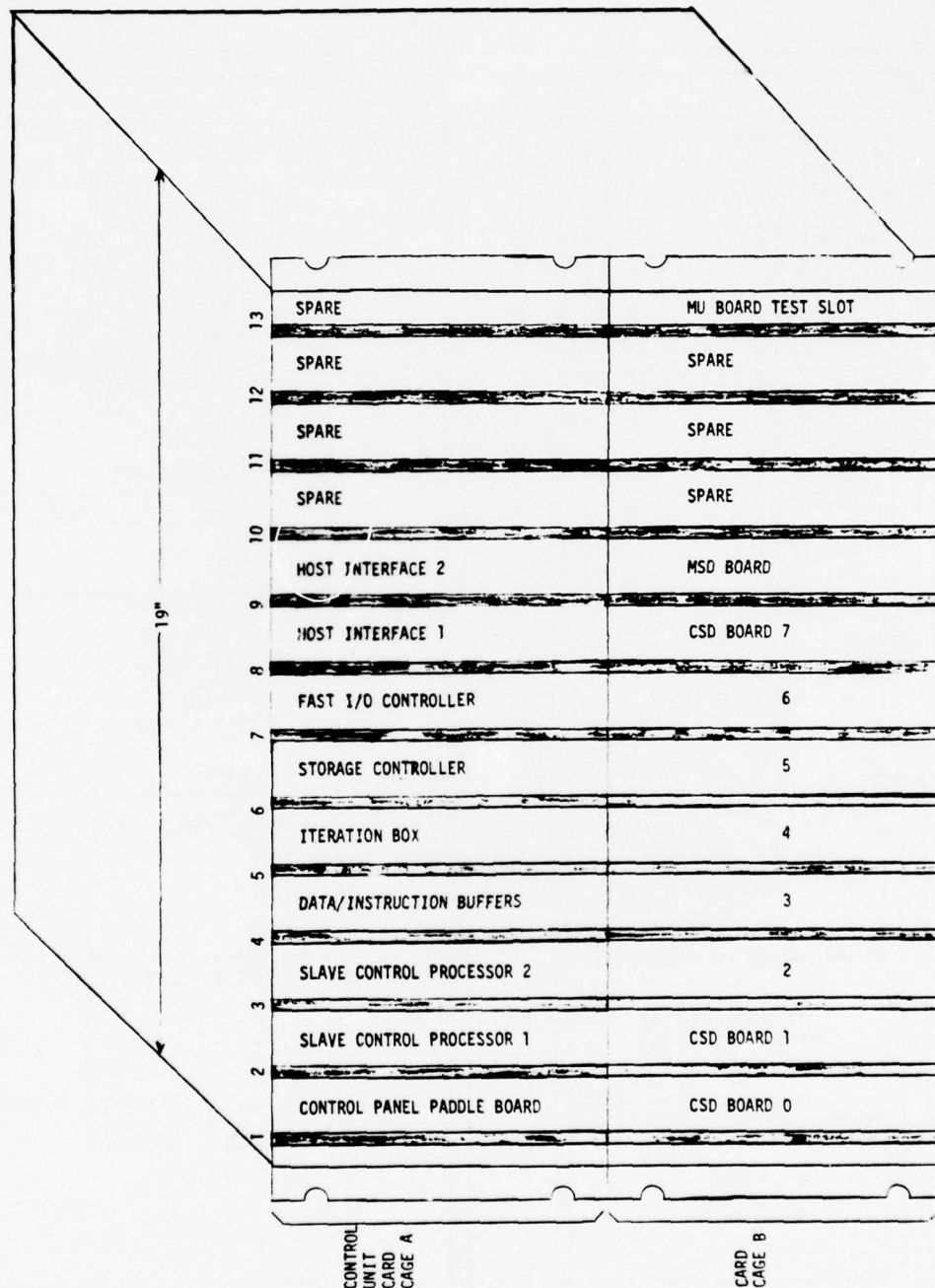


Figure 129. Control Unit Card Assignments

● Fast I/O controller	1
● Data/instruction buffers	1
● Iteration controller	1
● Control panel interconnect paddle board	1
	<hr/>
Subtotal	18

Additional card slots will be available for:

● Memory Unit board test	1
● Spare	7
	<hr/>
Total	26

Each of the cabinet signal distribution (CSD) boards will be capable of driving two Memory Unit cabinets (20 million bytes). Expansion to a 160-million byte system will require seven additional CSD boards. A functional block diagram of the CSD board is shown in Figure 130.

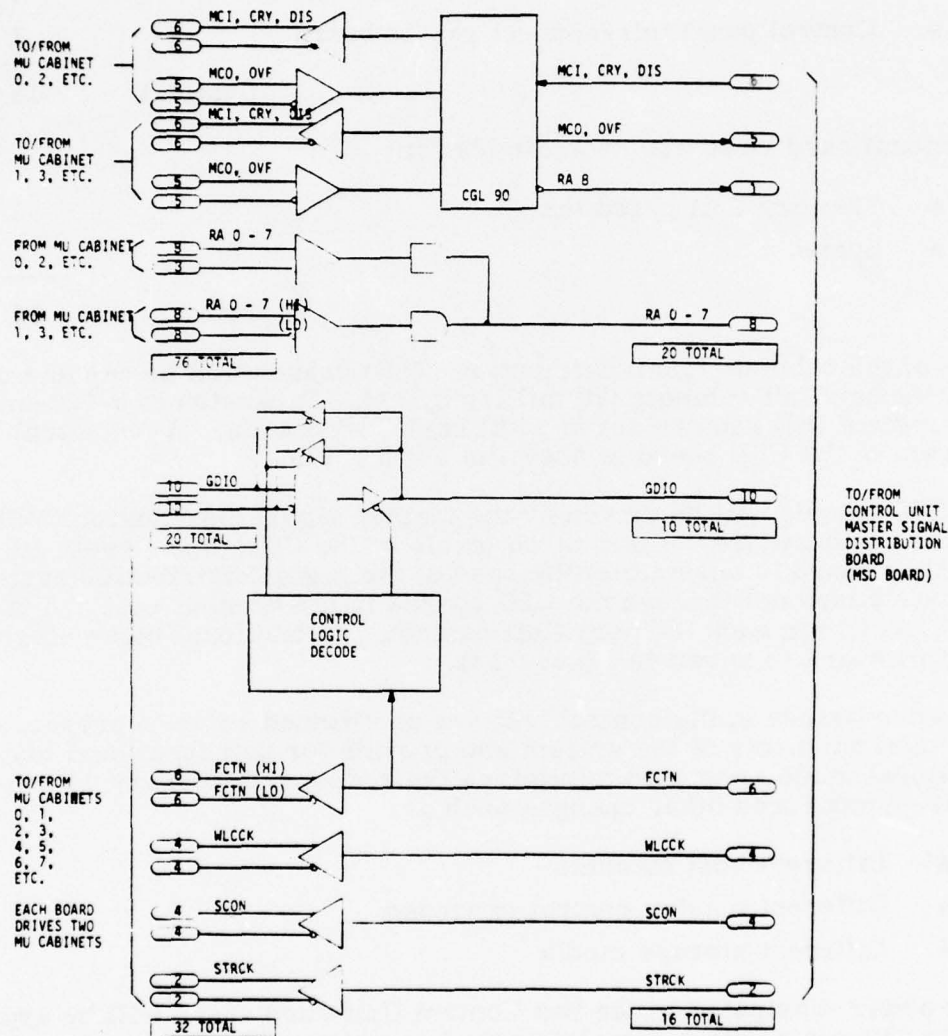
The CSD boards will be driven by the master signal distribution (MSD) board which will constitute the last three levels of the CGL tree (levels 10, 11, 12). The MSD board is essentially the base of the signal distribution system which ultimately fans out through the CSD boards in the Control Unit and the CBBs and SBBs within each Memory Unit cabinet. A functional block diagram of the MSD board is shown in Figure 131.

The other boards in the control unit are partitioned so as to preserve the functional autonomy of the system and provide for unit functional testing. Functional modularity also minimizes the redesign necessary when system requirements force other changes such as:

- Different host machine
- Different master control processor
- Different storage media

Total power dissipated in the two Control Unit card cages will be approximately 250 watts with a complete board complement. The master control processor will consume an additional 250 watts. Power supply efficiency of 50 percent will cause a requirement for 1 kilowatt of 115-volt, 60-cycle input power to the Control Unit cabinet.

LEVEL 9



TOTAL SIGNAL PINS REQUIRED	
SIGNAL	174
PMR/GND	6
SPARE @ 10%	20
TOTAL	= 200
APPROXIMATELY 4 WATTS/BOARD	

Figure 130. Control Unit Cabinet Signal Distribution Board

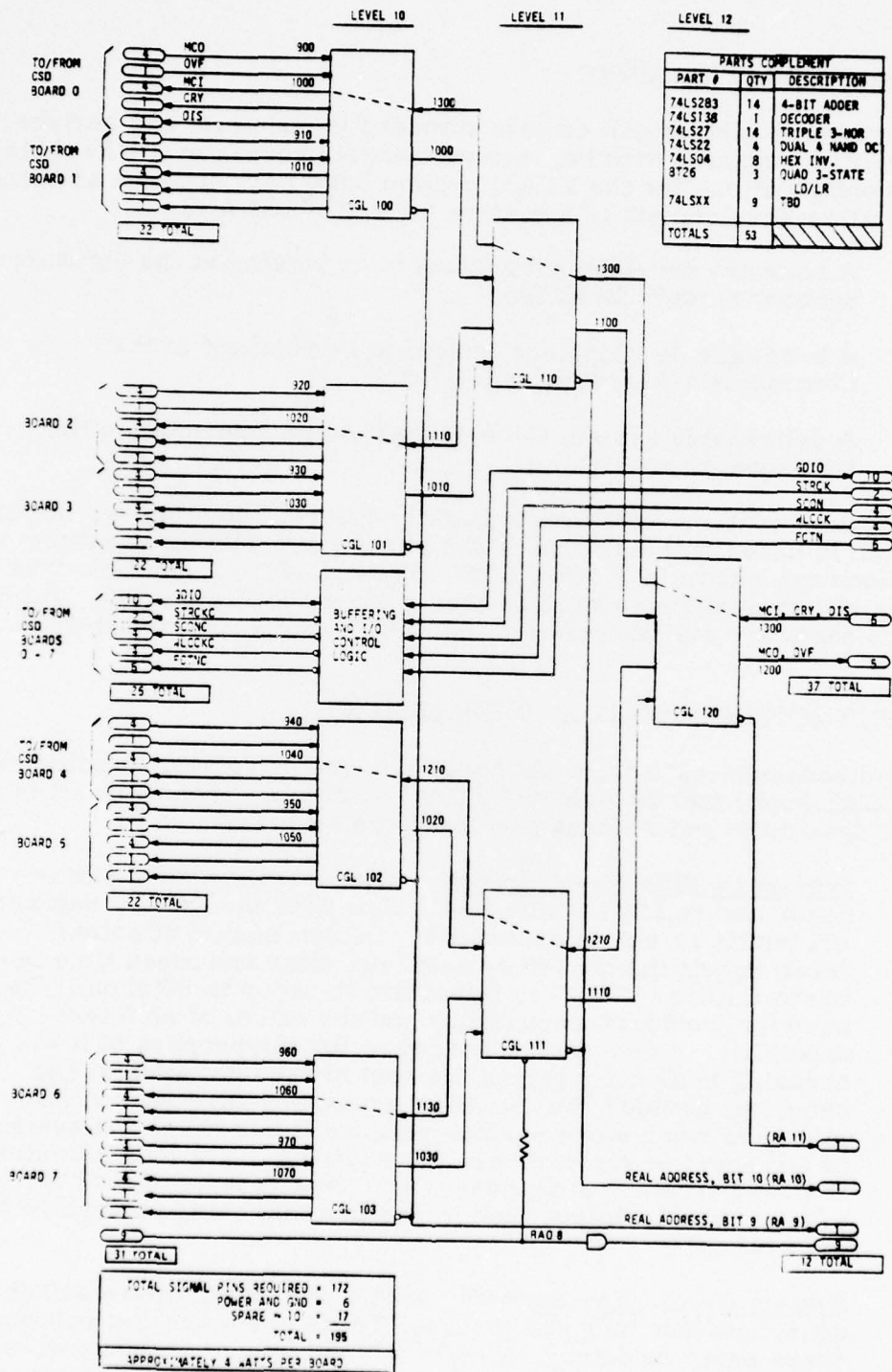


Figure 131. Master Signal Distribution Board

6.3 PERIPHERAL CABINET

The Peripheral cabinet will contain standard peripherals and peripheral controllers associated with the master control processor (PDP-11/45). Three configurations for the ECAM system have been outlined as being necessary to allow development of a system for SACWARDANS:

- A software development system to be located at the Software Subcontractor's facilities.
- A hardware development system to be retained at the Contractor's Aerospace Division.
- A deliverable system to be delivered and connected to the 6080 system at SAC.

Each of these systems will have a slightly different complement of peripherals which is based on the nature of system usage. The three system configurations are shown in Figures 132, 133, and 134. A detailed dimension drawing of a typical Peripheral cabinet is shown in Figure 135. Table 54 lists the hardware complement for the three systems as currently envisioned.

6.4 TEST METHODOLOGY AND EQUIPMENT

Before discussing test equipment hardware, we must first consider test and repair philosophy and the eventual ECAM system environment. It is necessary to determine and address requirements such as:

- System On-line Requirements - Can downtime for maintenance and repair be tolerated? How does the system degrade - gracefully or not so gracefully? Determination of actual (real) requirements for system "up" time and mean time between failure (MTBF) is important in order to determine the need for hardware redundancy and the extent of self-test capability. Unnecessary hardware for the purpose of increasing reliability raises the cost of the total system and should be avoided, but omitting necessary hardware to save money is much more serious because it can cause the system to fall short of performance expectations. The determination of what hardware is necessary and what is unnecessary is, to a large extent, determined by the function being performed by the system.
- System Repair Requirements - Who will do it, where will it be done, and how long can it take? These are questions relating to spare parts inventory, on-site test equipment, personnel, allowable mean time to repair (MTTR), the nature of line-replaceable units (LRUs) on-site, scheduling of maintenance and the proportion of it which is on-line, off-line, etc.

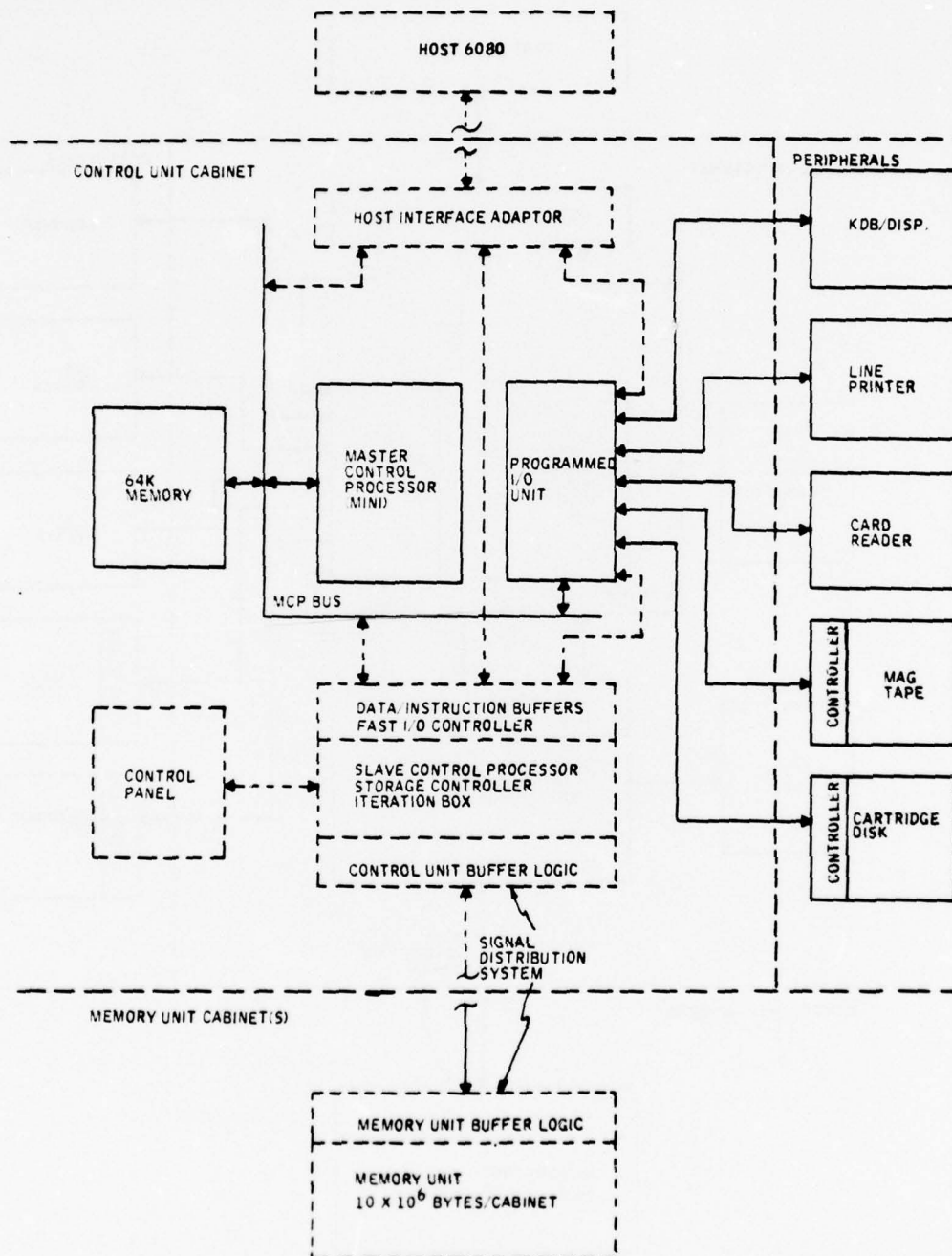


Figure 132. Software Subcontractor's Software Development Facility for ECAM

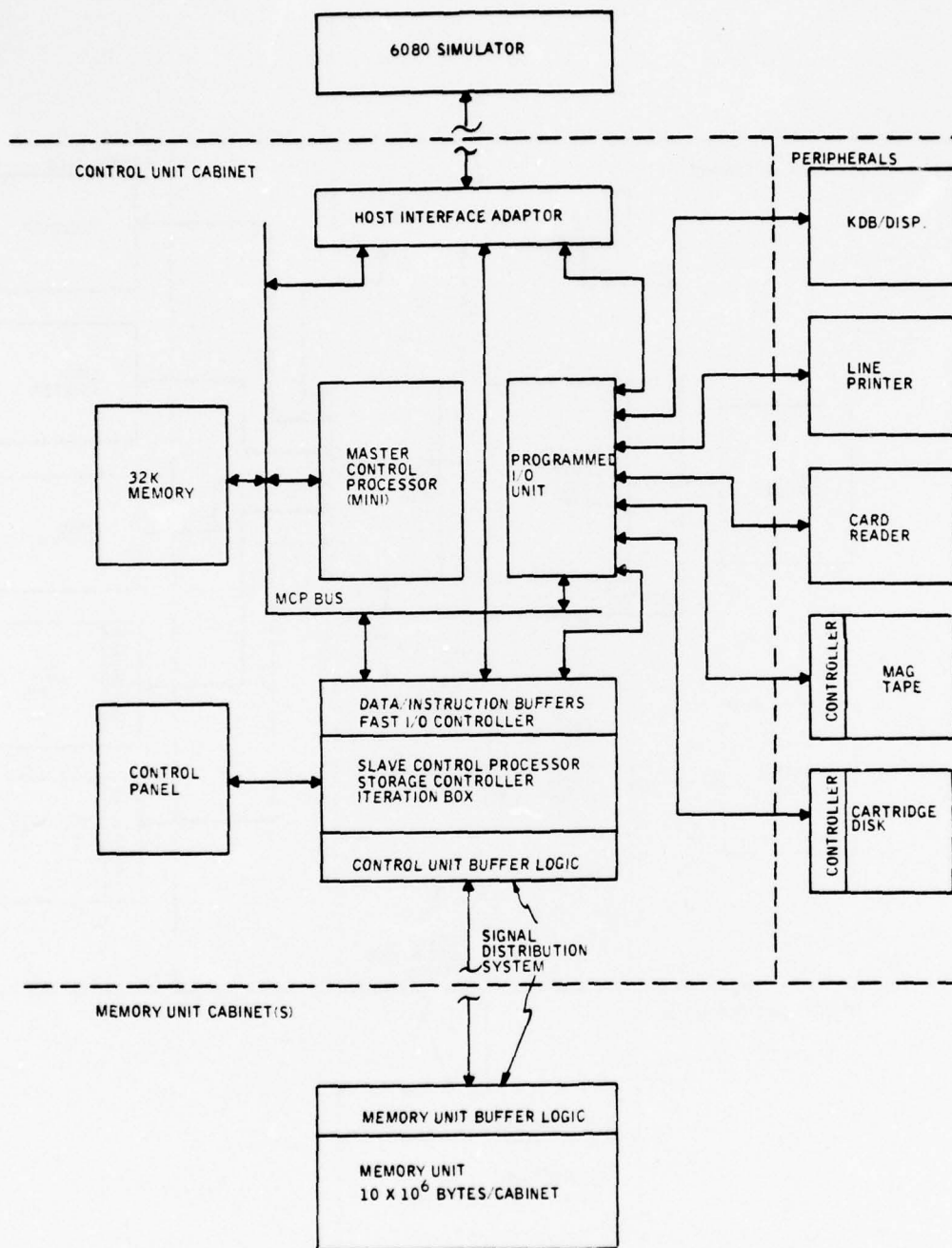


Figure 133. Aerospace Division SDF/Test Facility for ECAM

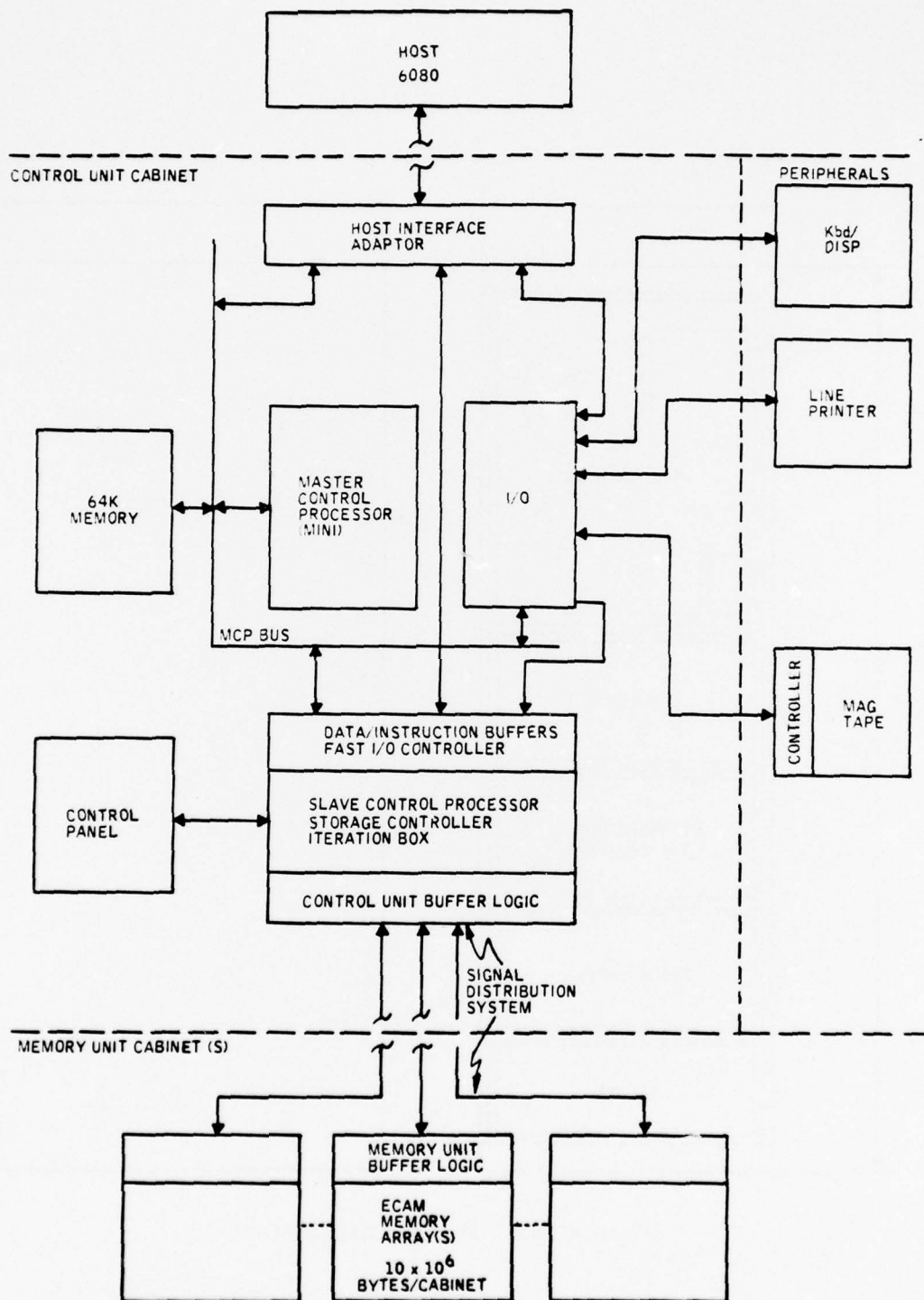


Figure 134. Deliverable ECAM System Functional Block Diagram

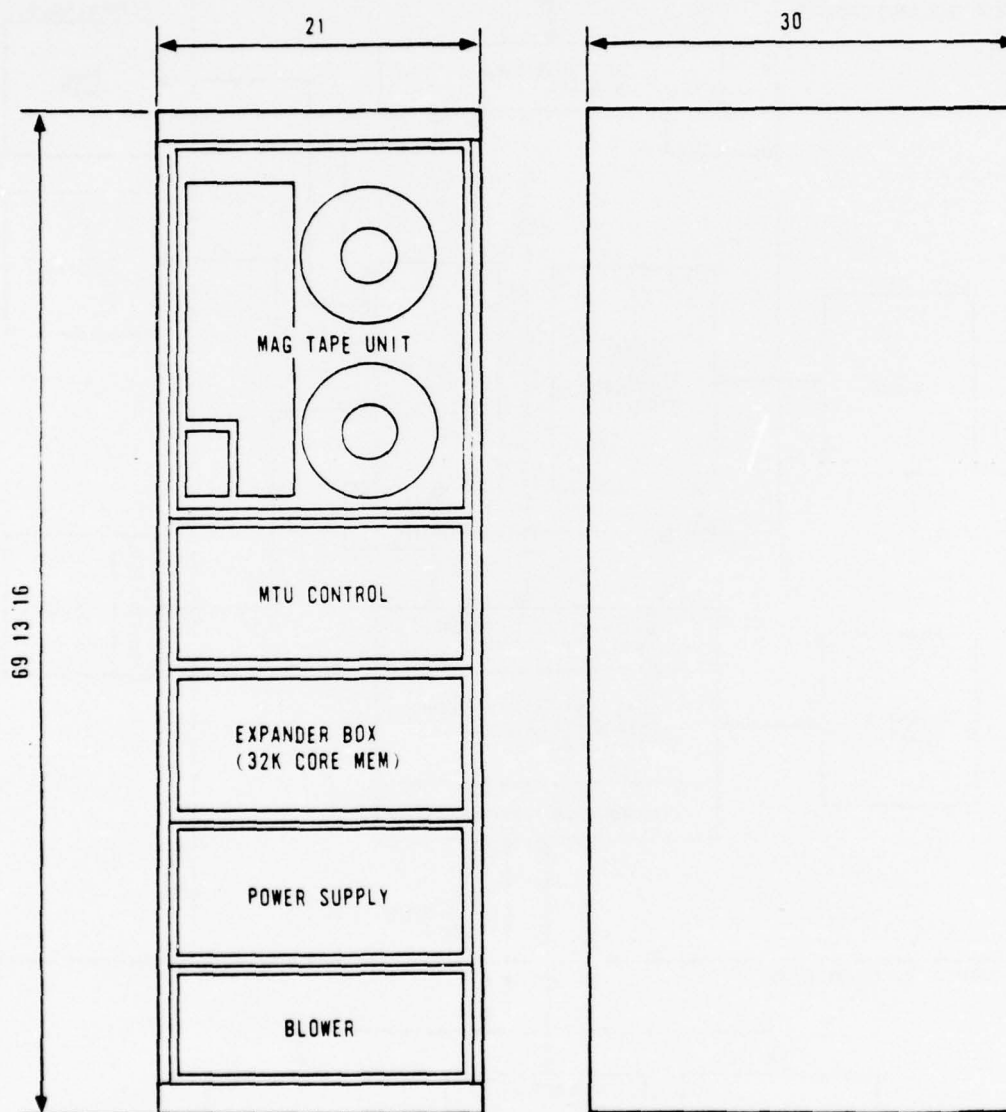


Figure 135. Peripheral Cabinet

Table 54. ECAM Minicomputer and Peripheral Requirements

Item	Quantity Required		
	Delivered System (64K)	Software Subcontractor SDF (64K)	Aerospace Division SDF/Test (32K)
<u>Main Frame:</u>			
1. PDP-11/45 FS; includes the following: a) Power fail/recovery b) 32K core memory with parity (980-nsec cycle) c) Memory interleave d) Cabinet e) 30 characters/sec KSR device and interface f) Hardware memory management	1	1	1
2. 32K core memory (MF-11UR) (with parity)	1	1	---
3. Programmable RTC (KW-11P)	1	1	1
4. Bootstrap (BM 873-YA)	1	1	1
<u>Peripherals/Accessories:</u>			
1. Card reader (CR-11); 300 cards/min.	--- ^a	1	1
2. KB/display (VT-52-AA); 24 lines, 80 columns	1	1	1
3. Line printer (LV-11BA); 500 LPM	1	1	1
4. Mag tape unit (TMA-11EA) with TU10 transport; 45 in/sec., 800 bpi	1	1	1
5. Cartridge disk (RK-11DE); includes cabinet, controller, 1 drive; 19M bits	--- ^a	1	1
6. Expansion box (BA-11K)	1	1	1
7. Peripheral mounting panel (DD-11A)	1	1	1
<u>Software:</u>			
1. Comprehensive software package (RSX-11M); supports the above configurations	1	1	1

^a Not required based on the assumptions that the delivered system will not be used as an SDF.

- System Failure Modes - This is simply a question of assigning responsibility for detecting failures and determining what should be done when one is detected. For example, power failure can be detected by the system or by an operator or both. If the system detects a power failure, it can notify the operator by some means while holding on with a backup system, or it can simply crash and let him figure it out. Usually, it is necessary to be able to have a rollback point to return to or be able to save critical data before power is lost completely, or both.

This study has not included a rigorous analysis of reliability requirements, but that which has been done allows us to prescribe the following baseline set of requirements for the hardware which can at least serve as a departure point for more in depth tradeoffs:

- System uptime should exceed 99 percent (23.75 hours per day).
- MTTR should be less than or equal to 5 minutes.
- Total sustained system failure is not acceptable.
- System hardware and software must provide assurance that there will be no loss of data in the event of failure.

These requirements dictate certain requirements for test and maintenance hardware: built-in and off-line. As such, the following capability will be provided at a minimum:

- On-line self-test and diagnostic software will be executed by the master control processor (MCP), in a background mode. The 6080 host will also execute background mode T&D software to verify the functional integrity of the master control processor and the host/MCP, I/O link.
- All tests involving the ECAM array will be conducted by the MCP. Illness in the MCP will imply a sick ECAM, with the possible exception that the host can force a system "dump" over the Fast I/O, DMA channel to its own memory.
- There will be no redundancy in the Control Unit. The assumption being that spare parts will be available on-site and that, in an operational mode, the LRUs are boards and replaceable modules which can be isolated and changed in 5 minutes or less. (This assumption is perhaps somewhat unrealistic, but then, so too also might be the requirement for 99 percent up-time.) Further, the system will be designed so as to allow removal and replacement of boards while the system is powered-up. An exception to this may be in the MCP commercial minicomputer which will have a separate power supply.

- The signal distribution system will be nonredundant. Spare cable sets for intercabinet interconnect and possibly even spare cabling harnesses for inside the Control Unit and Memory Unit cabinets will need to be available. This approach to minimizing repair time implies a modularity of wire harnesses and therefore additional connections and connectors which increase system cost. However, the alternative to rapid repair is full-up, standby redundancy which can automatically be switched in when failures occur. It is not obvious, at this point, that this type of failsafe hardware is required.
- The ECAM array is, by its very nature, fault-tolerant and redundant. Bad or faulty words can be isolated and blocked from participation in array activity. Memory words which go bad after having had information stored in them can be isolated and a backup copy retrieved from disk or tape and restored to another word. The assumption is that a nonvolatile backup system exists on-line and that all ECAM data are redundantly stored by either the MCP or the 6080 host. The MCP will have the capability to isolate blocks of words and relocate the contents to, in essence, create a hole in the ECAM array on which T&D operations can be performed. Faulty words found within a block being tested will have a dedicated "fault" flip/flop set which will block the word from further system activity. Fault counts and absolute address locations will be accomplished by copying the Fault flip-flops into the respective Match flip-flops and doing match counts, etc. Recordkeeping will be the responsibility of the MCP. Replacement of Memory Unit boards will require relocation of all information on the board by the Control Unit and disabling the board by either setting all Fault flip-flops, or alternatively a board Fault flip-flop. In either event, the MCP will notify the operator when a board can be removed. Such removal and replacement of Memory Unit boards will not interrupt normal system operation.
- The Memory Unit cabinet will have a board slot allocated for test capability. The functional operation of the test board will be to serve as a pseudo control unit which will be capable of exercising a memory board or group of boards in a predetermined sequence. This stand-alone capability will allow off-line testing and maintenance to be done without the need to tie up a Control Unit system to do Memory Unit board testing. Furthermore, it is possible that a direct link to the MCP will be developed to provide simple diagnostic capability without a slave control unit.

- The Control Unit cabinet will have at least one board slot allocated for a memory storage board and possibly two others for a type A and B Memory Unit buffer board to allow simulation of a Memory Unit cabinet within the Control Unit cabinet. If on-site repair capability is required, an off-line Control Unit with this capability should be all that will be required to accommodate all phases of system board test and repair.
- A control panel will be associated with the slave control processor (SCP) (see Figure 124) and interconnected with the system via a "paddle-board" technique which uses Card Slot 1 of Card Cage A in the Control Unit. This control panel will serve as a manual MCP with limited capability to control the SCP. The control panel will be detachable and removable and, as such, will only be used for factory checkout and field repair. A control panel will be installed in the off-line Control Unit which is used for repair and maintenance on-site.
- Additional, nondeliverable, test equipment which will have to be developed as part of a system development of this size is as follows:
 - A. CCD storage device test fixture for evaluation testing (1).
 - B. Hybrid subunit test fixtures (2).
 - C. Hybrid unit test fixture (1).
 - D. Word logic LSIC evaluation test fixtures (2).
 - E. Board test fixtures - it is assumed that computer-aided design (CAD) capability will be employed to do logic simulation, generate wire lists, and do loading and timing analysis. The logic design files developed will also be used to create board logic and continuity tests which will run on General Radio and DITMCO board testers. These two testers will require one fixture each to accommodate the ECAM boards.
 - F. 6080 simulator - This simulator will be a relatively unsophisticated hardware adapter which will be pin and function identical to the interface with the host 6080 machine. It is possible that another minicomputer will be used in conjunction with the simulator to perform certain 6080 operations and provide for complete functional testing of the host interface logic.

SECTION 7 PROGRAM PLAN

7.1 INTRODUCTION

This section describes a recommended follow-on course of action and the Contractor's plan for managing that course of action. The recommended program consists of two phases of activity which culminate in the delivery of the 10-million-byte ECAM system described in Section 6. Basically, the system consists of a Control Unit, one 10-million-byte Memory Unit, a complement of Peripheral Units, and an Operator's Station. (Refer to Figure 112.) In addition, a Software Development Facility (SDF), system test equipment, system software, and support software are also elements of the two-phase development program.

The program plan describes the way that the ECAM development would proceed if done by the Contractor. As such, it coordinates the activities of three entities: the Contractor's Systems and Research Center (S&RC), the Contractor's Aerospace Division, and an unspecified System Software Subcontractor. A detailed schedule and task plan for the S&RC and Aerospace Division activities are provided in Sections 7.2 and 7.3.

Section 7.4 contains a suggested Statement of Work (SOW) for the System Software Subcontractor. It covers only those tasks of the subcontract which directly affect the hardware. Additional, application-oriented tasks will also be required once a specific application and host are chosen. The Contractor believes that such tasks would be most efficiently handled under the same contract, but use of a separate contract and/or contractor is an option that is available to the Government.

7.2 SCHEDULE

The proposed ECAM program is, in essence, one 36-month program. However, it is separated into two phases because of the unique nature of the two gross types of activities associated with the program. The schedules for each phase are described in the following paragraphs.

7.2.1 Phase I Schedule

Phase I of the program involves the definition and conceptual design of the ECAM system. In addition, the development of certain test equipment and two specific long lead items, the hybrid and WLLC, are also initiated during Phase I. Those task elements are shown in the Phase I Master Schedule, Figure 136. Although the Phase I effort embodies a larger number of tasks,

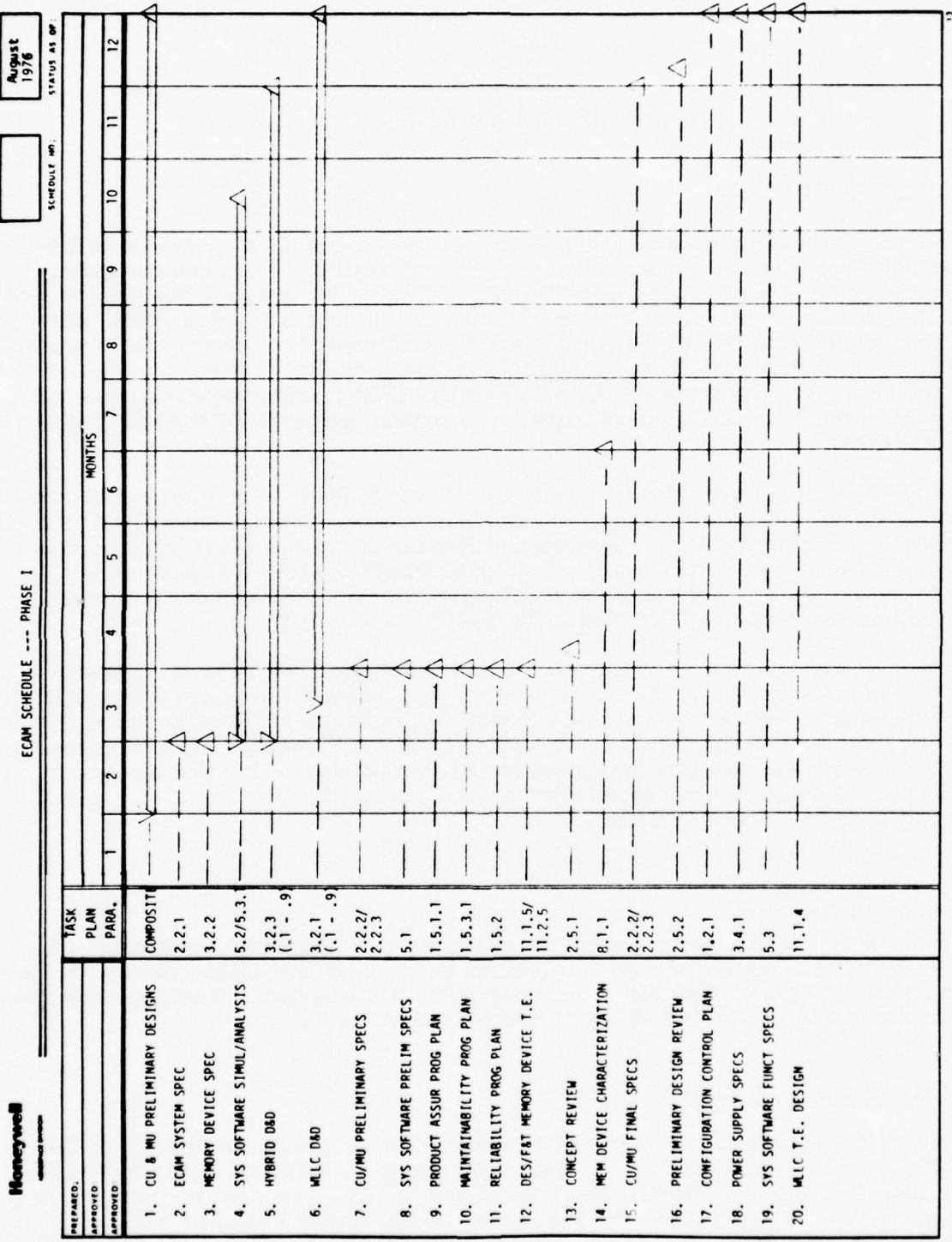


Figure 136. ECAM Phase I Master Schedule

only the major tasks are noted in Figure 136. A more detailed schedule delineating all Phase I tasks would be prepared prior to program go-ahead. The column on the schedule listing the Task Plan paragraph numbers is in reference to the Task Plan described in Section 7.3.

7.2.2 Phase II Schedule

Figure 137 is the Master Schedule for Phase II of the ECAM program. Along with the detail mechanical, electrical, software, and remaining test equipment designs, the development of the hybrid and WLLC will be completed under this phase. A major procurement activity will be undertaken to procure parts and equipment for the build and support of the 10-million-byte ECAM system which will be delivered at the completion of Phase II.

Similar to the Phase I effort, the Phase II tasks shown in Figure 137 are principal tasks and are representative of only a small portion of the total number of tasks. Prior to program go-ahead, a more detailed Phase II schedule will be prepared to afford finer control and monitoring of the program. The column on the schedule listing the Task Plan paragraph numbers is in reference to the Task Plan described in Section 7.3.

7.3 DETAIL HARDWARE DEVELOPMENT TASK PLAN

The total set of tasks for the ECAM program is divided into 11 major tasks. Each major task is further broken down into subtasks, sub-subtasks, etc.

The charts that follow (Figures 138 through 149) show the task plan organization. Figure 138 shows the overall categorization of the 11 major tasks; Figures 139 through 149 describe each of the major tasks in detail.

Following the task charts is a Task Plan describing each of the subtasks in detail for Tasks 1.0 (Program Management), 2.0 (System Design), 3.0 (Electrical Design), 4.0 (Mechanical Design), 6.0 (Support Software), and 8.0 (Test). Task 5.0 will be described in detail in the form of a Statement of Work to the System Software Subcontractor. Detailed Task Plan sheets will be prepared for the remaining four major tasks on an as-needed basis.

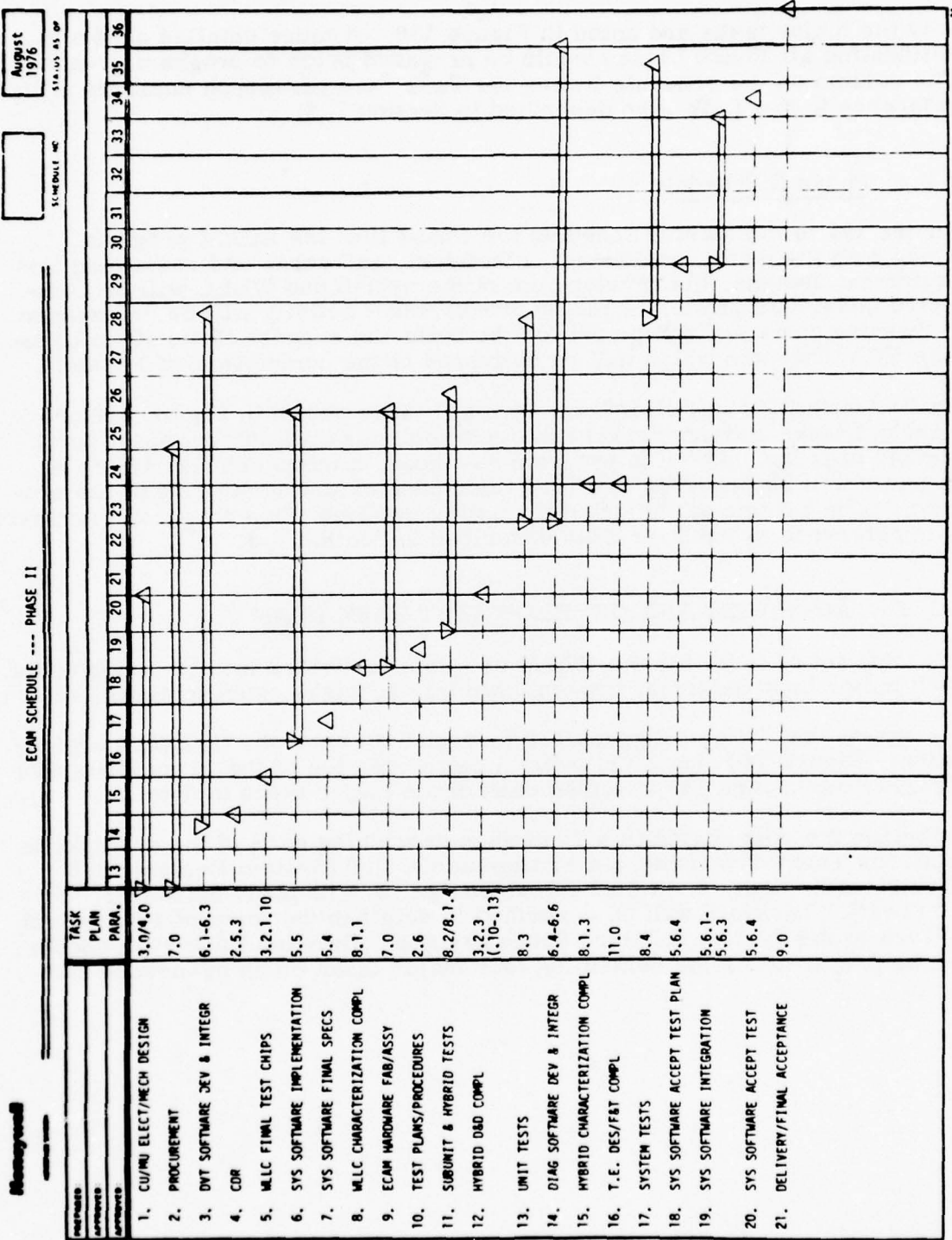


Figure 137. ECAM Phase II Master Schedule

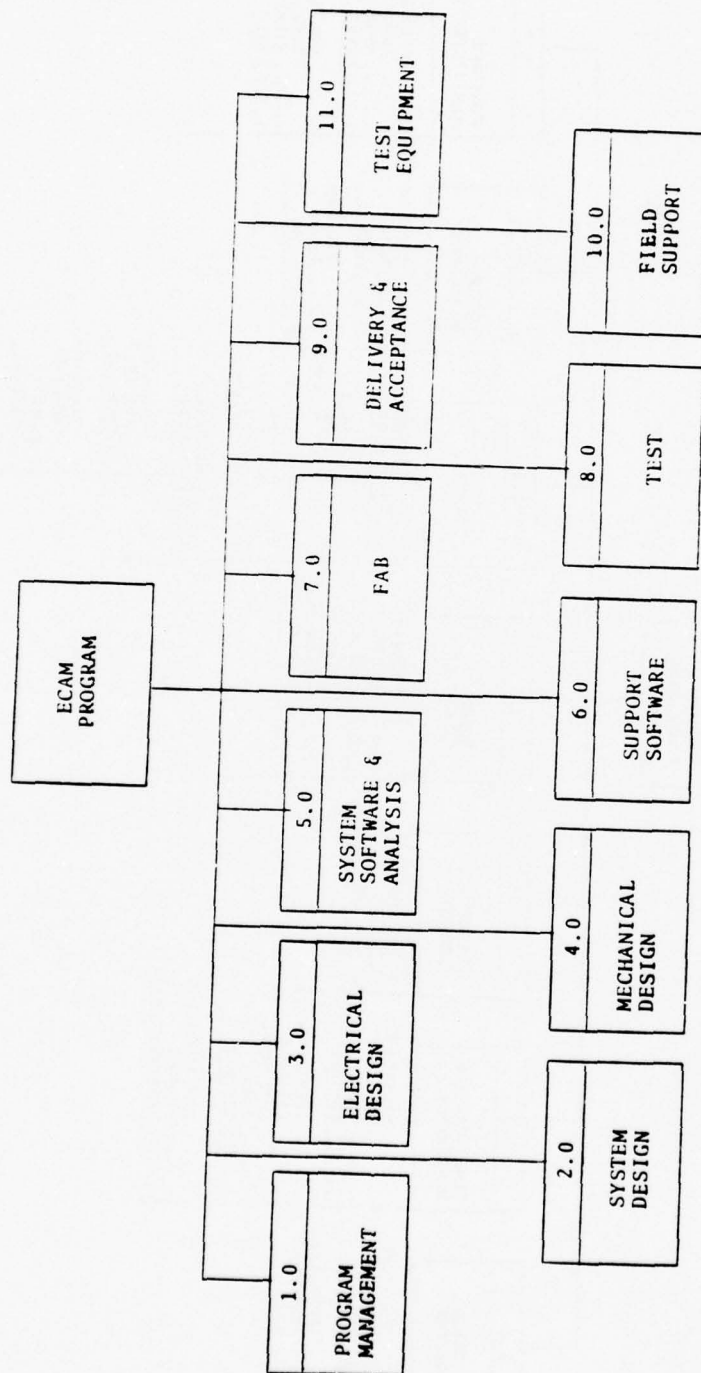


Figure 138. ECAM Program Overall Task Breakdown

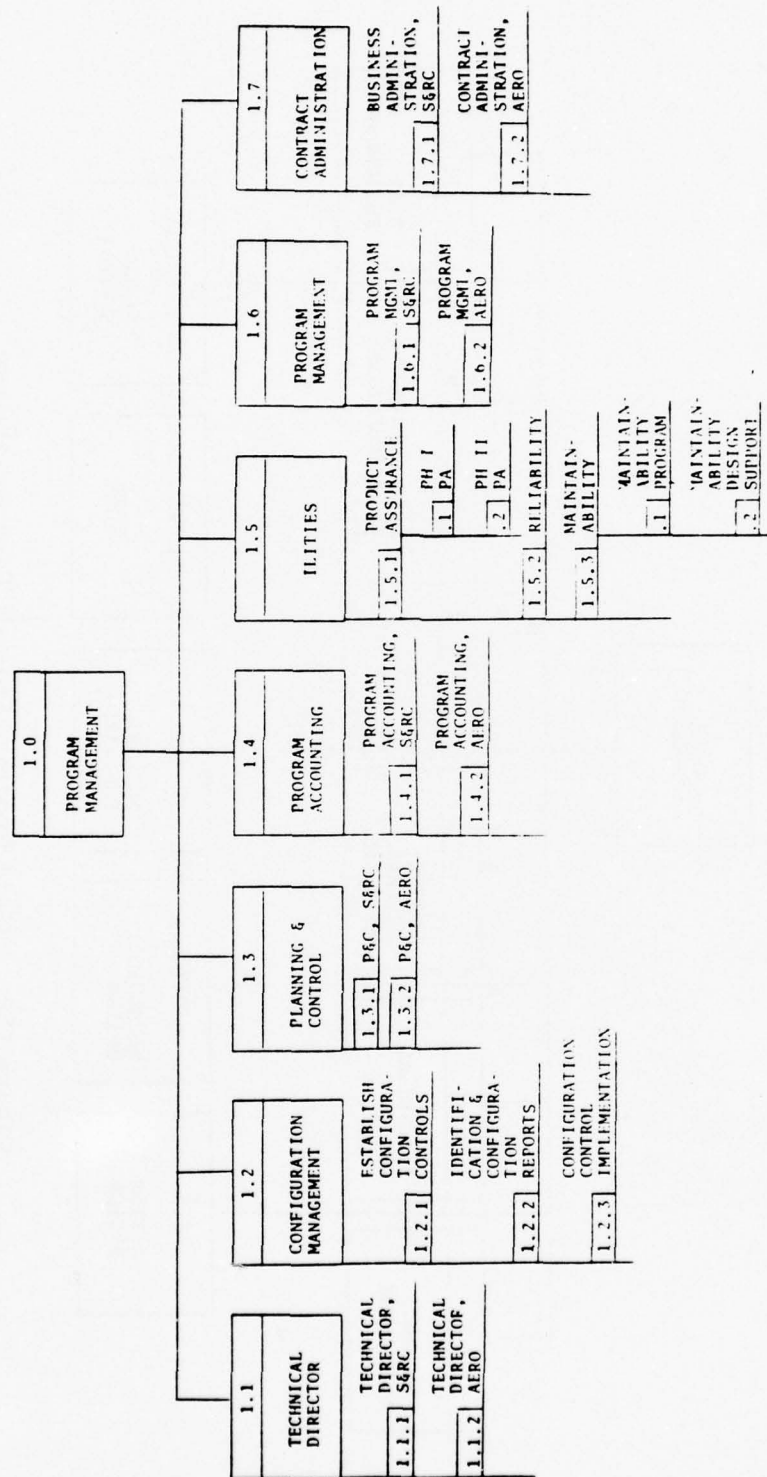


Figure 139. Task 1.0 Breakdown -- Program Management

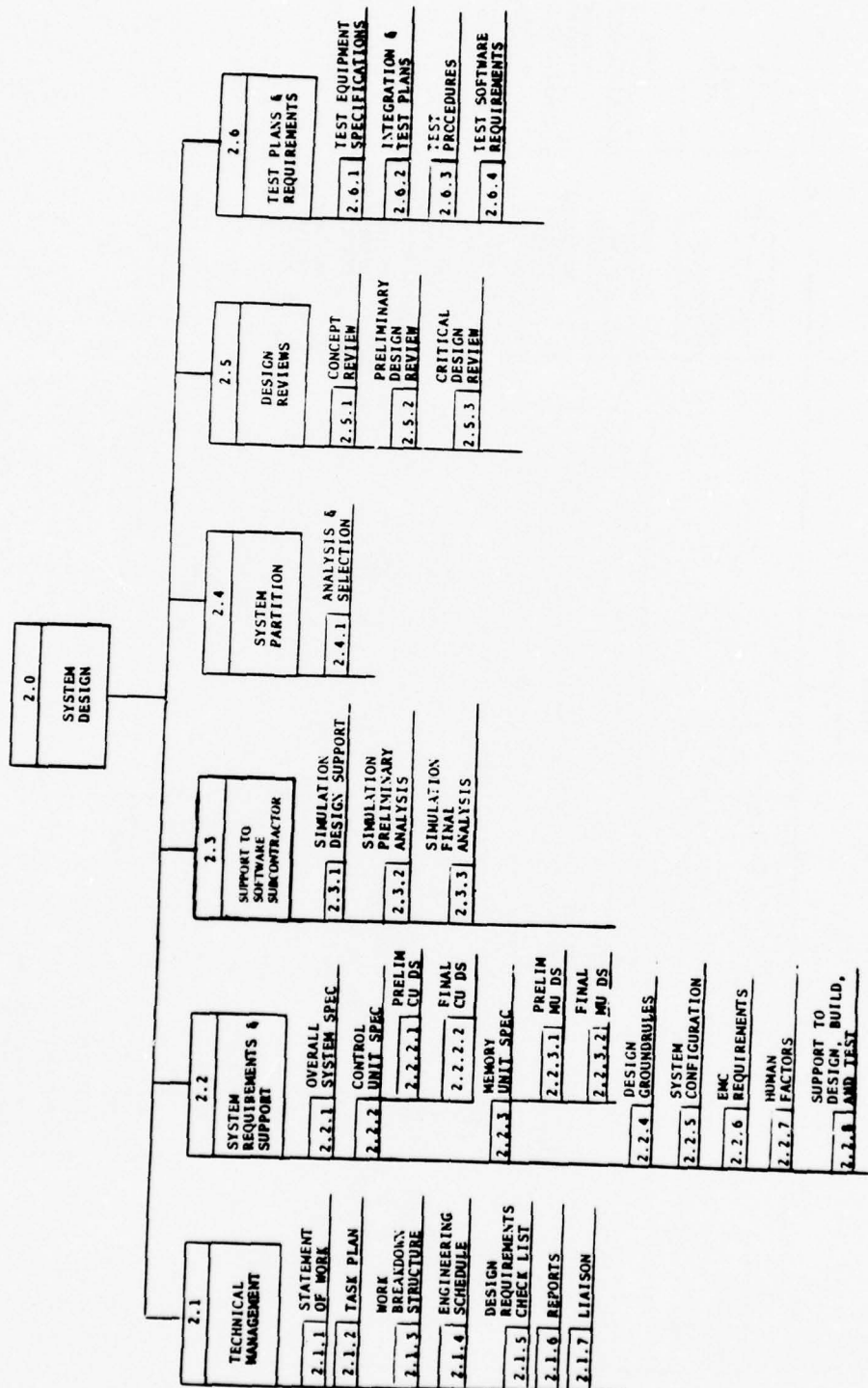


Figure 140. Task 2.0 Breakdown -- System Design

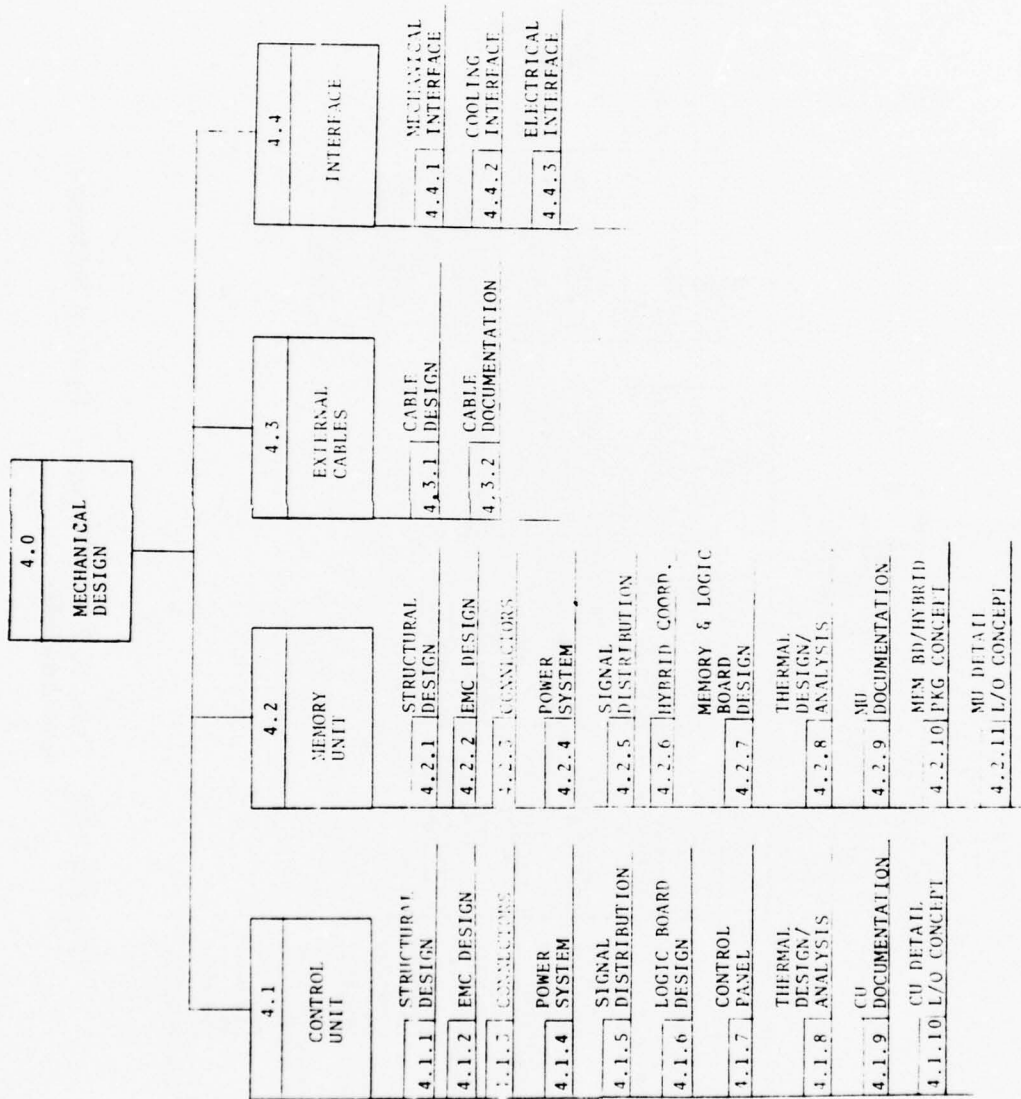


Figure 142. Task 4.0 Breakdown -- Mechanical Design

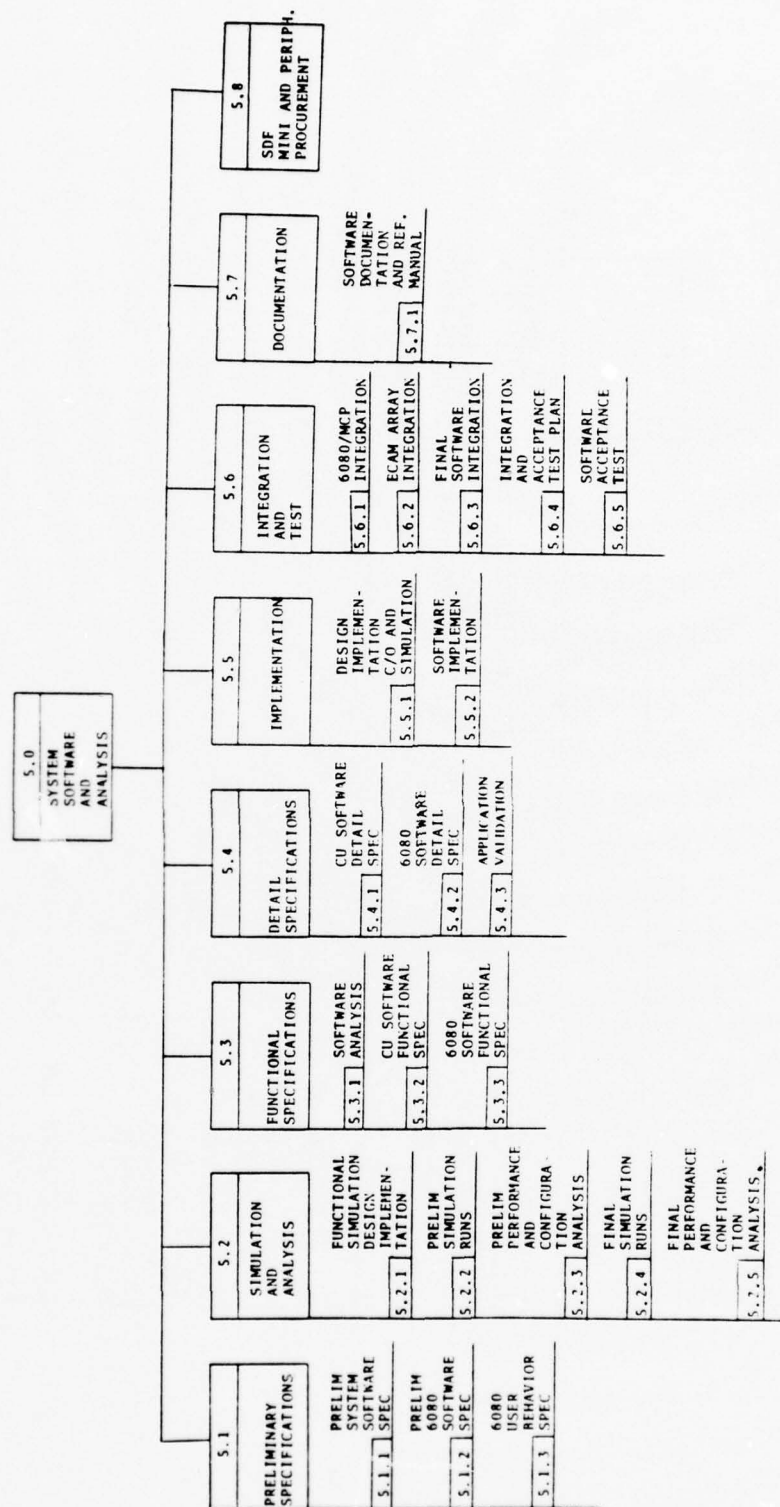


Figure 143. Task 5.0 Breakdown -- System Software and Analysis

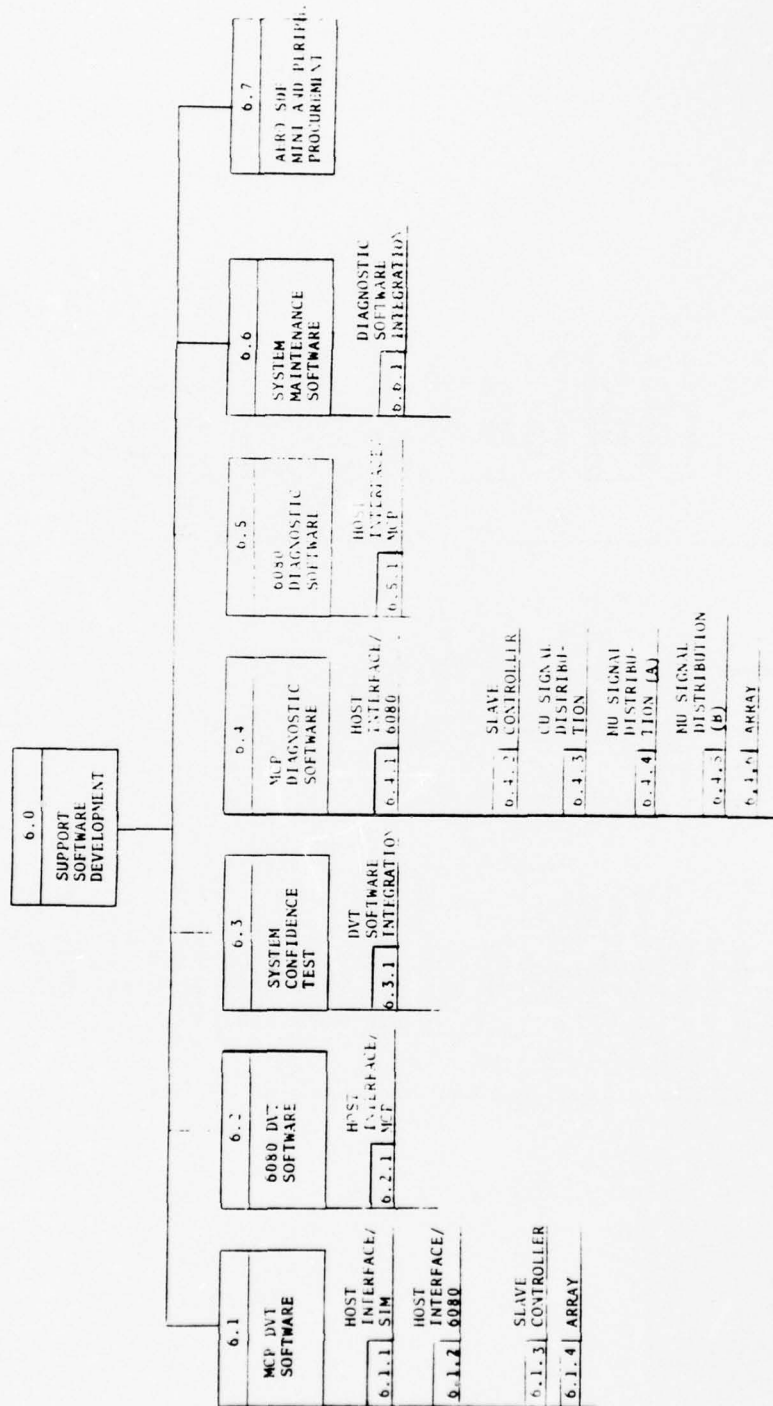


Figure 144. Task 6.0 Breakdown -- Support Software Development

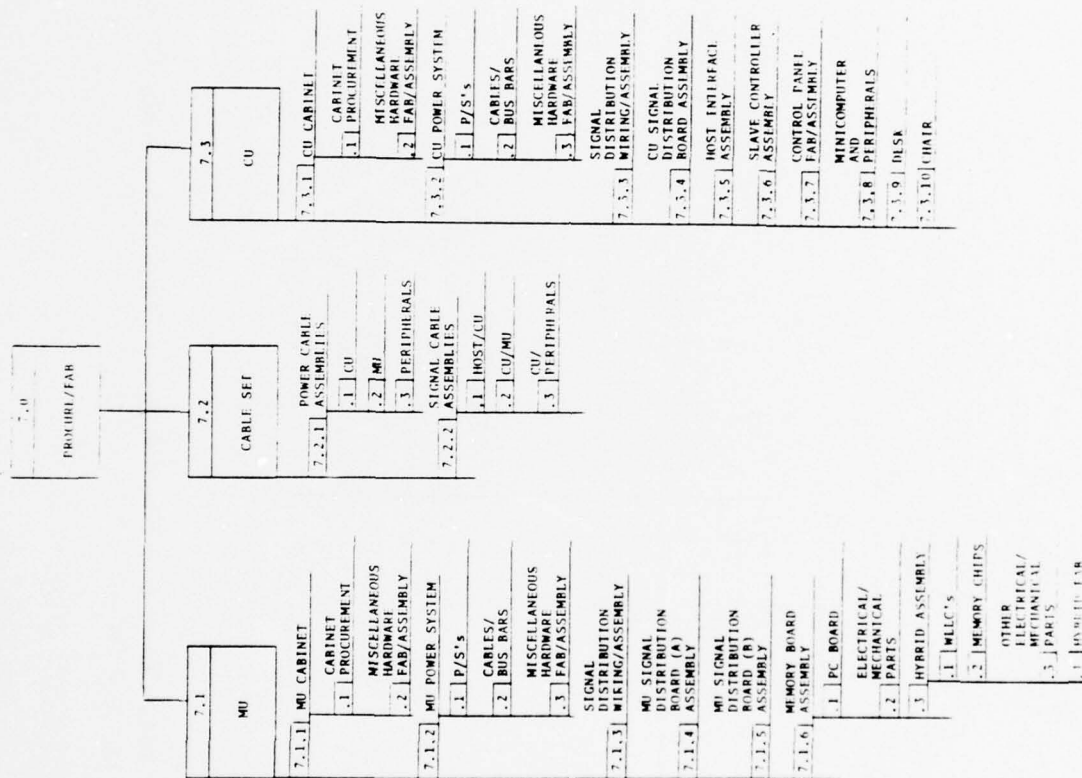


Figure 145. Task 7.0 Breakdown -- Procure/Fab

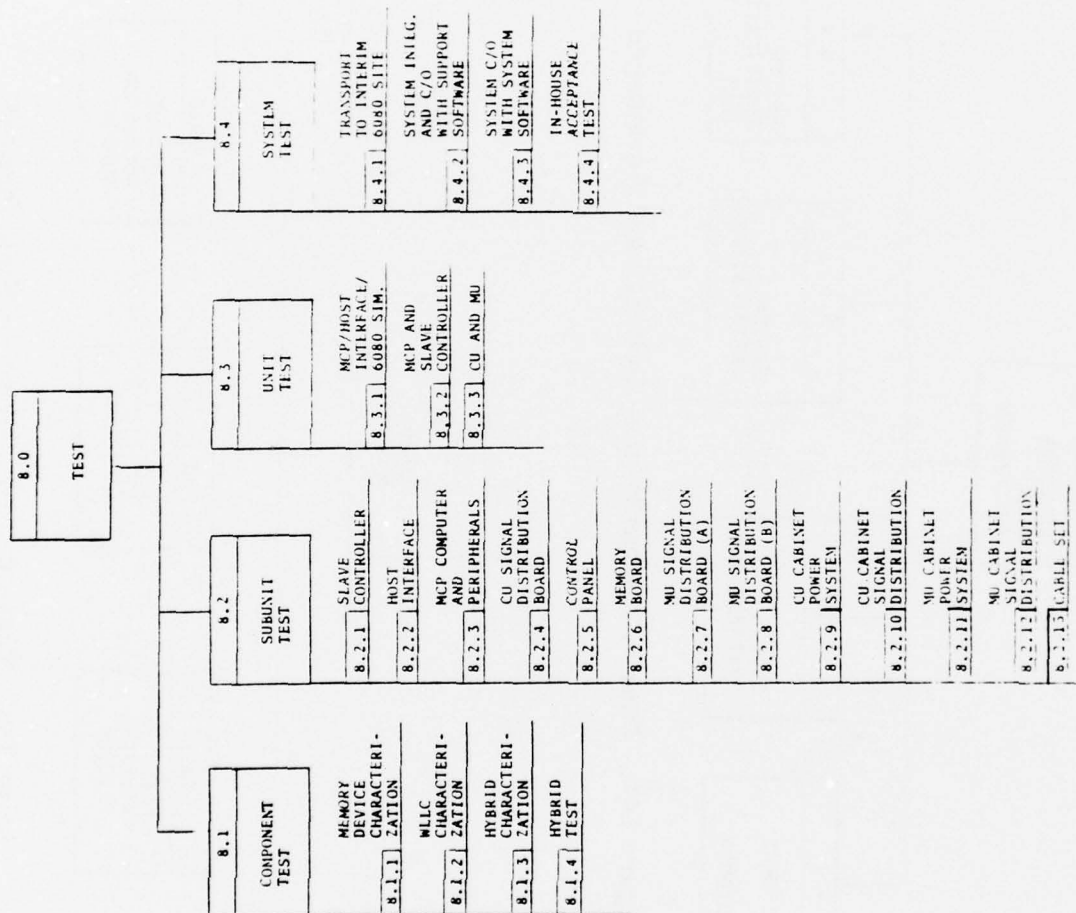


Figure 146. Task 8.0 Breakdown -- Test

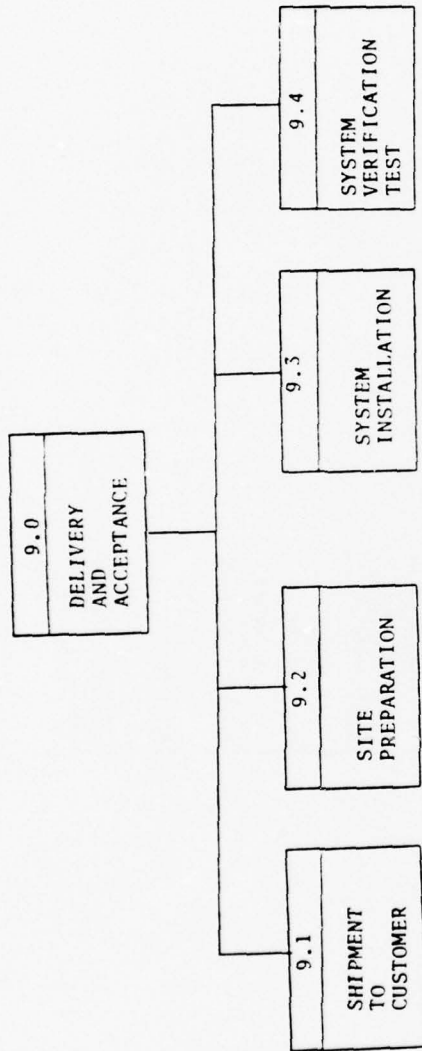


Figure 147. Task 9.0 Breakdown -- Delivery and Acceptance

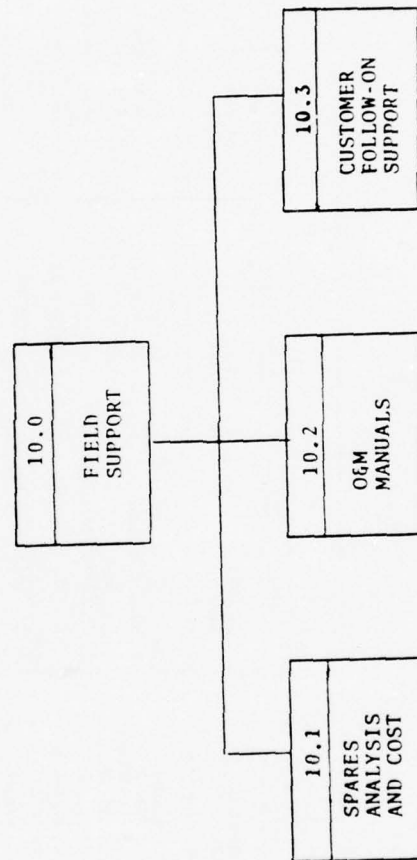


Figure 148. Task 10.0 Breakdown -- Field Support

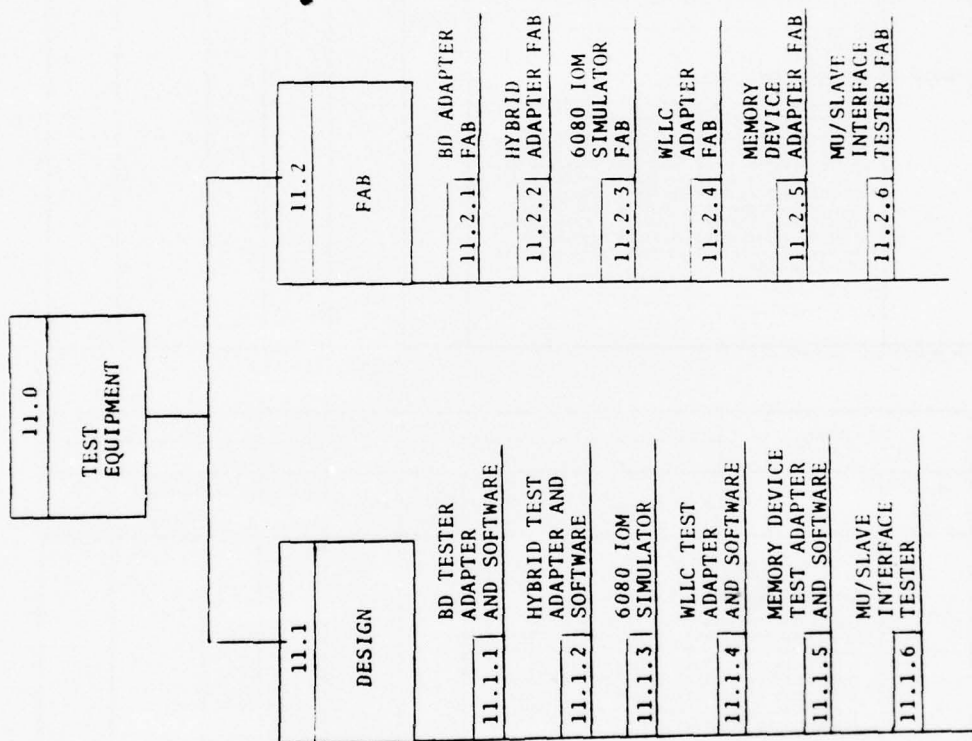


Figure 149. Task 11.0 Breakdown -- Test Equipment

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		PROJECT AUTHORITY NUMBER	
										B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER							
1.0	<u>PROGRAM MANAGEMENT</u>												
1.1	<u>TECHNICAL DIRECTOR</u>												
1.1.1	<u>Technical Director, S&RC</u> This task covers all activity associated with overall technical supervision of the ECAM development activity (including Aero Division and Software Subcontractor activities) during Phase I of the program. The Technical Director will be continually aware of the technical progress and will assure that contract technical requirements are met. During Phase II of the program, the S&RC Technical Director's scope of supervision shall be limited to the specific Phase II program tasks assigned to S&RC.												
1.1.2	<u>Technical Director, Aero</u> This task covers all activity associated with overall technical supervision of the ECAM development activity (including S&RC and Informatics activities) during Phase II of the program. The Technical Director will be continually aware of the technical progress and will assure that contract technical requirements are met. During Phase I of the program, the Aero Technical Director's scope of supervision shall be limited to the specific Phase I program tasks assigned to Aero Division.												

FORM NO. 10-11-00 11/10/61

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		PROJECT AUTHORITY NUMBER	
										B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER							
1.2	<u>CONFIGURATION MANAGEMENT</u>												
1.2.1	<u>Establish Configuration Controls</u> Configuration Management shall provide to all levels of management the necessary procedures and disciplines to achieve effective configuration control. This control shall be implemented on a continuing basis from initial definition of the ECAM system to the final use of the specific hardware and software, and shall be provided by: A. A Configuration Identification Baseline System which defines, through specifications and associated data, the requirements for all end items. B. A Configuration Control System which controls all changes to the end items. C. A Configuration Accounting System which documents all changes to baseline configurations.												
1.2.2	<u>Identification and Configuration Reports</u> Provide controls to assure proper hardware and software identification and Part No. changes in accordance with appropriate MIL-STD's and Honeywell design procedures. Assist in defining nameplate requirements and obtaining the "Request for Nomenclature" identification. Provide the Configuration Management reports (CIR, Bill of Material, etc.) necessary to document all engineering and "build to" requirements.												

FORM NO. 10-11-00 11/10/61

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER		PAGE		OF		
										B				August, 1976		
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER										
1.2.3	<u>Configuration Control Implementation</u> Provide configuration control on a continuing basis from the initiation of Phase II to final delivery of the software and hardware.															
1.3	PLANNING AND CONTROL															
1.3.1	<u>Planning and Control, S&RC</u> This task shall include all functions necessary to provide visibility of program progress to the customer and to assure cost and schedule adherence. The effort shall include: <ul style="list-style-type: none"> Generate/maintain Work Breakdown Structure (WBS). Prepare/coordinate customer/management reports. Operation of formal internal cost/schedule planning and control system. Prepare/maintain a program master schedule. 															
1.3.2	<u>Planning and Control, Aero</u> This task shall include all functions necessary to provide visibility of program progress to the customer and to assure cost and schedule adherence. The effort shall include:															

FORM NO. 10-PT-004 11/10/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER		PAGE		OF		
										B				August, 1976		
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER										
1.3.2	<u>Planning and Control, Aero</u> (Continued) <ul style="list-style-type: none"> Generate/maintain Work Breakdown Structure (WBS). Prepare/coordinate customer/management reports. Operation of formal internal cost/schedule planning and control system. Prepare/maintain a program master schedule. 															
1.4	PROGRAM ACCOUNTING															
1.4.1	<u>Program Accounting, S&RC</u> The Program Accountant shall monitor, analyze, and report the financial status of the program to the Program Manager. In addition, the Program Accountant shall prepare accurate and timely customer financial reports.															
1.4.2	<u>Program Accounting, Aero</u> The Program Accountant shall monitor, analyze, and report the financial status of the program to the Program Manager. The program billed and unbilled receivable balances shall be monitored to insure maximum collection of monies due. In addition, the Program Accountant shall prepare accurate and timely customer financial reports.															

FORM NO. 10-PT-004 11/10/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
							B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
1.5	ILITIES								
1.5.1	<u>Product Assurance</u>								
1.5.1.1	<u>Phase I F.A.</u> Develop a Product Assurance Program Plan addressing the following Quality Assurance elements as appropriate to the ECAM Program. <ul style="list-style-type: none"> Receiving Inspection Assembly Inspection and Process Control Development Quality Engineering Quality Assurance Data Provide assistance in the preparation of the System Specifications to insure that the Quality Assurance provisions are adequately covered.								
1.5.1.2	<u>Phase II F.A.</u> Implement the Product Assurance Program Plan which was prepared under Task 1.5.1.1.								

FORM NO. 74-11-004 11/20/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
							B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
1.5.2	<u>Reliability</u> Develop a Reliability Program Plan addressing the following Reliability Engineering elements. Conduct a Reliability Program in accordance with the approved plan. <ul style="list-style-type: none"> Reliability Predictions and Analyses Parts Reliability Program (LSIC and Hybrids) Failure Analysis Design Review Support - Participation Deliverable Software Validation (Minimum Standard Program) 								
1.5.3	<u>Maintainability</u>								
1.5.3.1	<u>Maintainability Program</u> Establish a minimum, but adequate, M Program. Provide assistance in the preparation of the System Specifications to insure that the maintainability provisions are adequately covered.								
1.5.3.2	<u>Maintainability Design Support</u> Implement the M Program which was defined under Task 1.5.3.1. Provide assistance to the design tasks to insure that the ECAM system design provides for rapid trouble-shooting and replacement of critical items. Perform trade-off studies as required and select optimum approaches to insure that the system maintenance requirements will be achieved. As part of this effort, appropriate M design criteria shall be formulated and coordinated with the design team.								

FORM NO. 74-11-004 11/20/62

TASK PLAN CONTINUATION SHEET		DATE		DATE		DATE		DATE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
1.6	PROGRAM MANAGEMENT								
1.6.1	Program Management, S&RC	<p>Assume the overall Program Management leadership role during Phase I of the ECAM Program. During Phase II, the scope of Program Management activities shall be limited to the specific S&RC tasks. As a minimum, the Program Management function shall be responsible for the following:</p> <p>A. Management of the program to assure performance against all contractual requirements.</p> <p>B. Attain program financial goals and objectives.</p> <p>C. Report to S&RC Division Management on conduct and status of the program.</p> <p>D. Define, analyze, and disseminate customer contract requirements.</p> <p>E. Establish and maintain a master program schedule.</p> <p>F. Provide program direction to the Technical Director, Planning and Control, Program Accounting, and Contract Administration functions.</p>							
1.6.2	Program Management, Aero	<p>Assume the overall Program Management leadership role during Phase II of the ECAM Program. During Phase I, the scope of Program Management</p>							

FORM NO. 74-PT-004 11/26/68

TASK PLAN CONTINUATION SHEET		DATE		DATE		DATE		DATE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
1.6.2	Program Management, Aero (Continued)	<p>activities shall be limited to the specific Aero tasks. As a minimum, the Program Management function shall be responsible for the following:</p> <p>A. Management of the program to assure performance against all contractual requirements.</p> <p>B. Attain program financial goals and objectives.</p> <p>C. Report to Aero Division Management on conduct and status of the program.</p> <p>D. Define, analyze and disseminate customer contract requirements.</p> <p>E. Establish and maintain a master program schedule.</p> <p>F. Provide program direction to the Technical Director, Configuration Management, Planning and Control, Program Accounting, Facilities, Contract Administration, Procurement and Production, Test Equipment, Field Support, System Delivery and Acceptance functions.</p>							

FORM NO. 74-PT-004 11/26/68

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		PAGE		OF	
												B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER									
1.7	CONTRACT ADMINISTRATION														
1.7.1	<u>Business Administration, S&RC</u> The Business Administrator shall interpret all aspects of the program contract and disseminate those interpretations in appropriate form to the affected departments. In addition, the following functions shall be provided: A. Conduct all contractual matters with Software Subcontractor (Phase I only) and Aero. B. Monitor all activities to insure that contractual obligations are being met. C. Negotiate all contractual matters with Software Subcontractor (Phase I only) and Aero, including funding. D. Initiate the contract release to carry out contracted work. E. Review subcontract documents to insure their compliance with all terms and conditions of the primary contract. F. Assure that the customer has continuous and proper assessment of program status, accomplishments, problems and plans.														
1.7.2	<u>Contract Administration, Aero</u> The Contract Administrator shall interpret all aspects of the program contract and disseminate those interpretations in appropriate form to the affected departments. In addition,														

FORM NO. PG-11-00a 11/28/63

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		PAGE		OF	
												B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER									
1.7.2	<u>Contract Administration, Aero (Continued)</u> the following functions shall be provided: A. Conduct all contractual communications with Software Subcontractor (Phase II only) and S&RC. B. Monitor all activities to insure that contractual obligations are being met. C. Negotiate all contractual matters with Software Subcontractor (Phase II only) and S&RC, including funding. D. Initiate the contract release to carry out contracted work. E. Review subcontract documents to insure their compliance with all terms and conditions of the primary contract. F. Assure that the customer has continuous and proper assessment of program status, accomplishments, problems and plans.														

FORM NO. PG-11-00a 11/28/63

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
						B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
2.0	<u>SYSTEM DESIGN</u>							
2.1	TECHNICAL MANAGEMENT							
2.1.1	<u>Statement of Work</u> Prepare the ECAM Phase I and Phase II Statements of work (SOWs). The SOWs shall serve as the instruments which define the services, materials, facilities and equipment to be furnished by Aero to the customer within the agreed-on schedules and requirement constraints. Preparation of the SOWs shall be coordinated with the Aero ECAM Program Manager.							
2.1.2	<u>Task Plan</u> Based on the contractual negotiations with the customer, update the Preliminary Engineering Task Plan (which was prepared prior to contract award) as required to reflect the mutually agreed on program tasks. The updated plan shall be defined as the Final Task plan and shall serve as the basis for all subsequent engineering effort. It shall be updated as required pending program changes.							

FORM NO. 74-PT-004 11/26/66

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
						B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
2.1.3	<u>Work Breakdown Structure</u> Based on the results of the contractual negotiations with the customer, update the Preliminary WBS (which was prepared prior to contract award) as required to reflect the mutually agreed on program effort. The updated WBS shall be known as the Final WBS and shall define the responsibilities and magnitudes of all program tasks. It shall be updated as required pending program changes.							
2.1.4	<u>Engineering Schedule</u> The Preliminary Engineering Schedule (which was prepared prior to contract award) shall be modified to agree with the Final Task Plan changes as a result of contractual negotiations with the customer. This schedule shall be updated as required pending program changes.							
2.1.5	<u>Design Requirements Check List</u> Update, as required, the Design Requirements Check List (DRCL) (which was originally prepared prior to contract award) to agree with the negotiated contract. The DRCL shall cover design data, procedural and documentation requirements for this effort as defined in Procedure 1.1 of the Design Procedures Manual. The DRCL shall be updated as required during the course of the program to agree with program changes.							

FORM NO. 74-PT-004 11/26/66

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.W. NUMBER	DATE	PAGE	OF
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER
2.1.6	<u>Reports</u> Prepare monthly status reports relating program progress for submittal to customer for review. These reports are to relate progress against the engineering schedule and relate problem or potential problem areas which could impact schedule. This task will also include preparing other customer reports and data items as defined upon contract award.					
2.1.7	<u>Customer Liaison</u> Perform customer coordination and liaison on a periodic basis to appraise the customer of program progress and developments. Any customer comments and change directives shall be coordinated to allow modifications to the task plans, WBS, and DRCL.					

FORM NO. 74-PT-001 11/28/62

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.W. NUMBER	DATE	PAGE	OF
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER
2.1	SYSTEM REQUIREMENTS & SUPPORT					
2.1.1	<u>Overall System Specification</u> Prepare the Part 1 "System level" Detail Specification (DS YGXXXXXX) for the overall ECAM system in accordance with Aerospace Division Design Procedure 2.1. This shall be a released document under I.D. control. The DS shall reflect the technical and system requirements as mutually agreed on by Aero and the customer during contract negotiations.					
2.1.2	<u>Control Unit Specification</u>					
2.1.2.1	<u>Preliminary CU DS</u> Prepare a "preliminary" version of the Part 1 "System level" Detail Specification (DS YGXXXXXX) for the Control Unit (CU) in accordance with Aerospace Division Design Procedure 2.1. The CU DS shall be in harmony with the requirements specified in the Overall System Specification (DS YGXXXXXX), the Design Groundrules as defined in task 2.1.4, and the System Functional Configuration as defined in task 2.1.5. The CU DS shall be updated as required to reflect the latest system requirements. The CU DS shall incorporate the detail requirements for the following major CU subfunctions:					

FORM NO. 74-PT-001 11/28/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER		PAGE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARA. NUMBER		PROJECT AUTHORITY NUMBER	
2.2.2.1	<u>Preliminary CU DS (continued)</u> 1. Master Control 2. Slave Controller 3. Host Interface 4. CU Power System 5. CU Signal Distribution System 6. Control Panel Prepare the boiler plate for the Part II "System Level" Detail Specification (DS YGXXXXXX) for the CU in accordance with Aerospace Division Design Procedure 2.1. Details for the above functional entities will be furnished under the Task 3.0 detail design effort.												
2.2.2.2	<u>Final CU DS</u> Update and finalize the CU DS which was prepared under Task 2.2.2.1												
2.2.3	<u>Memory Unit Specification</u>												
2.2.3.1	<u>Preliminary MU DS</u> Prepare a "preliminary" version of the Part I "System Level" Detail Specification (DS YGXXXXXX) for the Memory Unit (MU) in accordance with Aerospace Design Procedure 2.1. The MU DS shall be in harmony with the requirements specified in the Overall System Specification (DS YGXXXXXX), the Design Groundrules as defined in Task 2.2.4, and the System Functional Configuration as defined in Task 2.2.5. The MU DS shall be updated as required to reflect the latest												

FORM NO. 74-PT-08A 11/18/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER		PAGE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARA. NUMBER		PROJECT AUTHORITY NUMBER	
2.2.3.1	<u>Preliminary MU DS (continued)</u> system requirements. The MU DS shall incorporate the detail requirements for the following major MU subfunctions: 1. Memory Board 1.1 Hybrid* 1.1.1 Memory Element 1.1.2 Word Logic LSIC* 2. MU Power System 3. MU Signal Distribution System * The level of detail information in the MU DS is limited to that necessary for adequately defining the Memory Board requirements. Detail Procurement/Development Specifications for the asterisked components are to be prepared under Task 3.2. Prepare the boiler plate for the Part II "System Level" Detail Specification (DS YGXXXXXX) for the MU in accordance with Aerospace Division Design Procedure 2.1. Details for the above functional entities will be furnished under the Task 3.0 detail design effort.												
2.2.3.2	<u>Final MU DS</u> Update and finalize the MU DS which was prepared under Task 2.2.3.1.												

FORM NO. 74-PT-08A 11/18/62

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.W. NUMBER	DATE	PAGE	OF
PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	DATE
					R	AUGUST, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER
2.2.4	<u>Design Groundrules</u> Prepare, update, and coordinate a Design Groundrules document which specifies the design groundrules to be observed in the course of all ECAM electrical and mechanical design activity. These groundrules shall extend to major subcontract items as applicable. The groundrules are to include, but not be limited to: <ol style="list-style-type: none"> 1. Approved Parts List (IC's, Xistors, Connectors, etc.) 2. Limitations on usage of circuit elements (fan-out, circuit delay, application, power dissipation, voltage limits, etc.) 3. On board point-to-point wiring rules. 4. Off-board point-to-point wiring rules. 5. Cabling guidelines. 6. Shielding rules. 7. Power distribution guidelines. 					

FORM NO. 74-PT-08-1 11/28/62

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.W. NUMBER	DATE	PAGE	OF
PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	DATE
					E	AUGUST, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER
2.2.4	<u>Design Groundrules (Cont'd)</u> <ol style="list-style-type: none"> 8. Materials and processes. 9. Thermal analysis/design guidelines. 10. Human factors. 11. EMC. 12. High voltage. 13. Metal finish. 14. Dissimilar metal protection. 15. PC boards. 					
2.2.5	<u>System Configuration</u> Prepare and update, as required, the three following documents which define the ECAM system configuration: <ol style="list-style-type: none"> 1. <u>GO/INTO Chart</u> This chart is to describe the hardware makeup of the ECAM system. 2. <u>System Functional Diagram</u> 3. <u>System Interconnect Diagram</u> This diagram is to show all cabling between the HOST, CU, and ME. All cables and connectors shall be identified on the diagram. 					

FORM NO. 74-PT-08-1 11/28/62

TASK PLAN CONTINUATION SHEET		DATE		APPROVED BY		DATE		PAGE OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
2.2.6	<p><u>EMC</u></p> <p>Review and define the EMC requirements for the entire ECAM system and the CU and MU in terms of the functional/operational requirements.</p> <p>Provide EMC support to the preparation of the Detail Specifications in Tasks 2.2.1, 2.2.2, and 2.2.3.</p> <p>Provide guidelines for the electrical/mechanical EMC design for integration into the Design Groundrules of Task 2.2.4. The guidelines in turn are to be followed in the selection of components and in the detail electrical/mechanical design tasks.</p> <p>This task also includes support to the individual detail electrical and mechanical design tasks as required to ensure incorporation of the EMC features which are required to meet equipment and EMI requirements. Included are the power interface specifications for the RFI filters, rack and cable shielding design, external interface design, and system cooling design factors.</p> <p>Review the designs of all vendor-furnished equipments for conformity to the EMC requirements. For those equipments not meeting the EMC requirements, define the means for achieving same through additional shielding, packaging, etc.</p>								

FORM NO. 74, PT-000 11/76/02

TASK PLAN CONTINUATION SHEET		DATE		APPROVED BY		DATE		PAGE OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
2.2.7	<p><u>Human Factors</u></p> <p>Human Factor engineering shall be performed on the system design to guarantee that there are no operational safety hazards to humans. In this task, the system operator interfaces shall be reviewed and analyzed to: 1) insure simplicity of human operation; and 2) obtain optimum interactive layout to achieve the most convenient operation while minimizing probability of operator error.</p>								
2.2.8	<p><u>Support to Design, Build and Test</u></p> <p>Provide support and coordination to the electrical and mechanical effort during the design, build and test activities. This support effort is to cover all activities relating to specifications, print releases, test procedures, and system build, checkout, and test. This task shall provide the effort necessary to resolve any conflicts between the system specification requirements and the actual design, build, and test activities.</p>								
2.3	<u>SUPPORT TO SOFTWARE SUBCONTRACTOR</u>								
2.3.1	<p><u>Simulation Design Support</u></p> <p>Provide support during the functional simulation design and implementation effort under Task 5.2.1. Provide CU and MU technical details to Informatics as required.</p>								

FORM NO. 74, PT-000 11/76/02

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
						B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
2.0	TEST PLANS & REQUIREMENTS							
2.0.1	<u>Test Equipment Specifications</u> Create the specifications necessary to define the equipment required to test and evaluate the system to determine that it functions to the requirements of the System Specifications. These specifications shall be written in accordance with standard Aerospace procedures.							
2.0.2	<u>Integration and Test Plans</u> Create the plans necessary to test the individual subsystem units and to integrate and test these units forming the final system configuration. These plans shall be written as defined by Aerospace Division Design Procedure No. 5.3.							
2.0.3	<u>Test Procedures</u> Prepare System and Subsystem Test Procedures to evaluate the functions of the system to determine that it functions as required by the System Specifications. The procedures shall be as defined by Aerospace Division Design Procedure No. 5.3. This task shall include the preparation of both pre-acceptance and final acceptance test procedures.							
2.0.4	<u>Test Software Requirements</u> Define the requirements necessary for the test software to evaluate the operation of the system and subsystem components. The specification for this software shall be as defined by Aerospace Division Design Procedure No. 5.3.							

FORM NO. 10, PT-004 11/26/62

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
						B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
3.0	<u>ELECTRICAL DESIGN</u>							
3.1	CONTROL UNIT							
3.1.1	<u>Host Interface</u>							
3.1.1.1	<u>Host Interface Circuit/Logic Design</u> In accordance with the CU DS (Host Interface section) and the Design Groundrules, perform the Host Interface detail logic and circuit designs; prepare schematics/logic diagrams, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Prepare the detail Host Interface section for the Part II CU DS prepared under Task 2.2.2.							
3.1.1.2	<u>Host Interface Simulation</u> Provide computer simulation support to Task 3.1.1.1. The simulation shall be used in the formulation and verification of the Host Interface logic structure, operation and algorithms.							
3.1.2	<u>Master Control</u> The Master Control consists of the Master Control Processor (MCP) and Main Memory (MM). Under this task, the detail electrical interfaces (timing, levels, pin assignment, etc.) shall be defined and documented for the MCP and MM as required to assist in the completion of other detail functional designs.							

FORM NO. 10, PT-004 11/26/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		PAGE		OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARAL. NUMBER		PROJECT AUTHORITY NUMBER			
3.1.3		<u>Slave Controller</u>													
3.1.3.1		<u>Slave Controller Circuit/Logic Design</u> The Slave Controller consists of basic control logic plus a semiconductor micro-program memory. In accordance with the CU DS (Slave Controller section) and the Design Groundrules, perform the Slave Controller detail logic, circuit and firmware designs, prepare schematics/logic diagrams and microcode, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Prepare the detail Slave Controller section for the Part II CU DS prepared under Task 2.2.2.													
3.1.3.2		<u>Slave Controller Simulation</u> Provide computer simulation support to Task 3.1.3.1. The simulation shall be used in the formulation and verification of the Slave Controller logic structure, operation and algorithms.													

FORM NO. 74-11-08 11/20/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		PAGE		OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARAL. NUMBER		PROJECT AUTHORITY NUMBER			
3.1.4		<u>Support To CU Circuit Board Design</u> Provide electrical design support to the general purpose circuit board design selection process under Task 4.1.6. Electrical requirements relating to power/ground distribution, signal distribution, power, etc. shall be defined under this task.													
3.1.5		<u>Control Panel</u> In accordance with the CU DS (Control Panel section) and the Design Groundrules, perform the Control Panel detail logic and circuit designs, prepare schematics/logic diagrams, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures and prepare a brief discourse on the theory of operation. Prepare the detail Control Panel section for the Part II CU DS prepared under Task 2.2.2.													

FORM NO. 74-11-08 11/20/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE	
		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	DATE	OF	
						B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
3.2	MEMORY UNIT								
3.2.1	<u>Word Logic LSIC Chip Development</u>								
3.2.1.1	<u>WLLC Detail Specification</u> Prepare a Part 1 "Device Level" Detail Specification (DS 34XXXXXX) for the Word Logic LSIC Chip (WLLC). The requirements stipulated in the MU DS for the Word Logic function shall be incorporated in the WLLC DS.								
3.2.1.2	<u>WLLC Design Groundrules</u> Define and document the standard cell library and design groundrules to be observed in the detail design of the WLLC.								
3.2.1.3	<u>WLLC Logic Design</u> Based on the WLLC DS and the WLLC Design Groundrules, perform the WLLC detail logic design, prepare schematics/logic diagrams, generate interconnect information, provide drafting coordination, review drafting generated documentation (schematics), prepare detail test procedures, prepare brief discourses on theory of operation. NOTE: This task shall be interactive with Task 3.2.1.4 and the detail design shall be defined as being complete after the simulation process indicates that all functional requirements of the WLLC are met.								

FORM NO. PG-PT-000 11/20/68

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE	
		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	DATE	OF	
						B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
3.2.1.4	<u>WLLC Simulation</u> Using the HISIM system or equivalent, simulate the WLLC to insure that all functional requirements are met. The simulation process shall check and verify all of the chip logic and timing functions. The effort under this task shall be interactive with the detail design activity under Task 3.2.1.3.								
3.2.1.5	<u>WLLC Layout</u> Using an interactive graphics system (Computer Vision, Kalma, etc.), perform the Word Logic chip layout. The generated composite check plots shall be analyzed for errors and corrected based on the schematics generated under Task 3.2.1.3.								

FORM NO. PG-PT-000 11/20/68

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE		OF	
PREPARED BY		DATE		APPROVED BY		DATE		EXTENSION NUMBER		DATE	
								B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER					
3.2.1.5	<u>WLLC Layout (Cont'd)</u> Upon finalizing the layout, the chip layout information shall be transferred to magnetic tape in a format compatible with the mask generation equipment.										
3.2.1.6	<u>WLLC Mask Generation</u> Through the use of Electromask equipment and using the magnetic tape containing the WLLC chip layout information (Task 3.2.1.5), generate 10X chrome or glass emulsion mask masters for the WLLC chip.										
3.2.1.7	<u>Process WLLC Test Chips</u> Using the 10X mask masters provided under Task 3.2.1.6, process a minimum of 10 WLLC chips. Perform a functional probe and test of the chips to verify that the chips meet all of the functional requirements as defined in the WLLC Detail Specification (DS 34XXXXXX). All chip design errors or discrepancies shall be noted and documented and shall serve as the basis for the corrective design activity under Task 3.2.1.8.										
3.2.1.8	<u>WLLC Corrective Design</u> Based on the errors and discrepancies noted under Task 3.2.1.7, modify the WLLC chip design to correct same. This task shall include re-simulation and re-layout of the chip as required. The WLLC DS and other documentation provided under Task 3.2.1.5 shall be updated as required. Based on the finalized design, prepare Part II for the WLLC DS.										

FORM NO. 74-PT-08a 11/20/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE		OF	
PREPARED BY		DATE		APPROVED BY		DATE		EXTENSION NUMBER		DATE	
								B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER					
3.2.1.9	<u>Finalized WLLC Mask Generation</u> Using the updated mag tape WLLC chip layout information from Task 3.2.1.8, generate finalized 10X chrome or glass emulsion mask masters for the WLLC chip.										
3.2.1.10	<u>Process Finalized WLLC Test Chips</u> Using the finalized 10X mask masters provided under Task 3.2.1.9, process a minimum of 120 WLLC chips. Perform a final functional probe and test of the chips to verify that the chips meet all of the functional requirements as defined in the WLLC Detail Specification (DS 34XXXXXX). The 120 chips will be allocated as follows: WLLC Chip Characterization - 10 Chips Char. Hybrid Assy & Characterization - 80 Chips Evaluation Hybrid - 16 Chips Contingency - 14 Chips										

FORM NO. 74-PT-08a 11/20/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE		OF	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER		DATE	
								B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER					
3.2.2	<u>Memory Element Specification</u> Define the development and production status of the various CCD (and other) memory element chips which are candidates for the ECAM application. Define their functional features characteristic and limitations. Select the optimum technology and chip which best suits the ECAM requirements (performance, cost, reliability, environmental, flexibility, etc.) Prepare a "Device Level" Detail Specification (DS 34XXXXXX) describing the memory chip. The requirements stipulated in the MU DS for the memory chip function shall be incorporated in the memory element DS. The DS shall serve as the memory chip procurement specification as well as for design requirements for the hybrid development effort under Task 3.2.3.										
3.2.3	<u>Hybrid Development</u>										
3.2.3.1	<u>Hybrid Detail Specification</u> Prepare a "Device Level" Detail Specification (DS 34XXXXXX) for the hybrid device. The requirements stipulated in the MU DS for the hybrid function shall be incorporated in the hybrid DS.										
3.2.3.2	<u>Hybrid Electrical Design</u> In accordance with the hybrid DS and the Design Groundrules, perform the hybrid detail logic and circuit designs, prepare schematics/logic diagrams, generate intra-hybrid and intra-board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare										

FORM NO. 14, PT-004 11/10/66

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE		OF	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER		DATE	
								B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER					
3.2.3.2	<u>Hybrid Electrical Design (Continued)</u> brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Prepare the hybrid section for the Part II MU DS prepared under Task 2.2.3.										
3.2.3.3	<u>Hybrid Simulation</u> Provide computer simulation support to Task 3.2.3.2. The simulation shall be used to verify that the hybrid logic/circuit design will meet all of its functional requirements.										
3.2.3.4	<u>Process Development</u> Based on the estimated 3"x3" hybrid, develop the necessary fabrication techniques for same. Also, define the techniques necessary to bond the larger chips, such as the memory chips, to the substrate or carriers. Document the results of this effort.										
3.2.3.5	<u>Establish Hybrid Layout Groundrules</u> Review the circuit design documentation provided under Task 3.2.3.2. Based on the electrical design requirements, establish the Hybrid Layout Groundrules. The Hybrid Layout Groundrules shall define such parameters as number of layers, signal line widths and spacing, etc.										

FORM NO. 14, PT-004 11/10/66

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	DELIVERABLE NUMBER	PAGE OF
						B	August, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
3.2.3.6	<u>Artwork Generation</u> Based on the schematics generated under Task 3.2.3.2 and the LSIC chip pin out and pad site information defined under Tasks 3.2.1 and 3.2.2, generate a first pass 10X drafting layout and perform a check on same. Digitize the corrected 10X drafting layout and generate a 10X pen plot. Check and correct the 10X pen plot. From the corrected 10X pen plot, generate a final 1X pen plot of the hybrid circuit paths.						
3.2.3.7	<u>Generate Assembly Drawings</u> Generate the assembly drawings and documentation required for the Hybrid device.						
3.2.3.8	<u>Generate Assembly Layout</u> Based on the fabrication techniques defined under Task 3.2.3.4 and the assembly drawings prepared under Task 3.2.3.7, generate an Assembly Layout for the Hybrid device. The Assembly Layout shall define the step-by-step procedure for fabrication, assembling, and inspecting the Hybrid.						
3.2.3.9	<u>Fabricate Evaluation Hybrid</u> Fabricate one (1) Evaluation model Hybrid for subsequent functional test purposes. Assembly of ICs on the hybrid substrate shall be accomplished under Task 3.2.3.10.						

FORM NO. 16-PT-004 11/28/62

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	DELIVERABLE NUMBER	PAGE OF
						B	August, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
3.2.3.10	<u>Assemble Evaluation Hybrid</u> Using the WLLC chips furnished under Task 3.2.1.10, and using the memory and MSI/MSI chips furnished under Tasks 7.1.6.3.2 and 7.1.6.3.3, assemble the evaluation hybrid which was previously fabricated under Task 3.2.3.9.						
3.2.3.11	<u>Test Evaluation Hybrid</u> Test the Evaluation Hybrid to verify that it meets all of the functional requirements as stipulated in the Hybrid DS which was prepared under Task 3.2.3.1. All errors and discrepancies shall be noted and documented.						
3.2.3.12	<u>Hybrid Corrective Design</u> Based on the design errors and discrepancies noted under Task 3.2.3.11, perform the necessary electrical design, and generate new artwork, assembly drawings, and an assembly layout. Document the updated design in finalized form. Based on the final design, prepare Part 11 for the Hybrid DS.						

FORM NO. 16-PT-004 11/28/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		DATE		PAGE OF		
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARA. NUMBER		PROJECT AUTHORITY NUMBER		DATE		
3.2.3.13		<u>Fabricate And Test Characterization Hybrids</u> Using the finalized Hybrid design defined under Task 3.2.3.12, fabricate and functionally test five (5) Hybrids for subsequent characterization test purposes. WLLC chips fabricated under Task 3.2.1.10 shall be used in the five hybrids. The memory and SSI/MSI chips provided under Tasks 7.1.6.3.2 and 7.1.6.3.3 shall be used in the five hybrids. The actual hybrid characterization tests shall be performed under Task 8.1.3.														
3.2.4		<u>Memory Board</u> In accordance with the MU DS (Memory Board section), the Design Groundrules, and the hybrid DS, perform the memory board detail logic and circuit designs; prepare schematics/logic diagrams, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Prepare the detail Memory Board section for the Part I MU DS prepared under Task 2.2.3.														

FORM NO. 74-PT-004 11/28/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVED BY		DATE		S.O.W. NUMBER		DATE		PAGE OF		
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARA. NUMBER		PROJECT AUTHORITY NUMBER		DATE		
3.3		<u>SIGNAL DISTRIBUTION</u>														
3.3.1		<u>CU Cabinet Internal Interconnects</u> In accordance with the signal distribution groundrules defined under Task 2.2.4 and based on the signal interconnect wire list information defined under Task 3.1 and 3.3.4, perform the detail logic signal interconnect (back plane) and cable designs for the CU cabinet circuit boards, control panel, mini-computer and main memory, and external connectors. This activity shall include the generation of all CU cabinet logic signal interconnect wire lists and cable drawings, coordination with drafting, review of drafting-generated documentation, and specification of connectors.														
3.3.2		<u>MU Cabinet Internal Interconnects</u> In accordance with the signal distribution groundrules defined under Task 2.2.4 and based on the signal interconnect wire list information defined under Tasks 3.2.4, 3.3.5 and 3.3.6 perform the detailed logic signal interconnect (back plane) and cable designs for the MU cabinet circuit boards and external connectors. This activity shall include the generation of all MU cabinet logic signal interconnect wire lists and cable drawings, coordination with drafting, review of drafting-generated documentation, and specification of connectors.														

FORM NO. 74-PT-004 11/28/62

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.D. NUMBER	DATE	PAGE	OF
PREPARED BY		DATE	APPROVED BY	DATE	ELICTION NUMBER	DATE
					B	AUGUST, 1970
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.D. PARA. NUMBER	PROJECT AUTHORITY NUMBER
3.3.3	<u>External Cables</u> In accordance with the signal distribution groundrules defined under Task 2.2.4 and based on the signal interconnect wire list information defined under Tasks 3.3.1 and 3.3.2, perform the detail design for all ECAM external logic signal interconnect cables (between the 6080 Host computer, CU cabinet, MU cabinet, and mini-computer/peripherals). This activity shall include the generation of cable drawings, coordination with drafting, review of drafting-generated documentation, specification of connectors, and coordination with the activity of Tasks 3.3.1 and 3.3.2.					
3.3.4	<u>CU Signal Distribution Board</u> In accordance with the CU DS (Signal Distribution System section) and the Design Groundrules, perform the CU Signal Distribution Board detail logic and circuit designs; prepare schematics/logic diagrams, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Prepare the detail CU Signal Distribution System section for the Part II CU DS prepared under Task 2.2.2.					

FORM NO. 16, PT-004 11/20/62

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.D. NUMBER	DATE	PAGE	OF
PREPARED BY		DATE	APPROVED BY	DATE	ELICTION NUMBER	DATE
					B	AUGUST, 1970
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.D. PARA. NUMBER	PROJECT AUTHORITY NUMBER
3.3.5	<u>MU Signal Distribution Board (A)</u> In accordance with the MU DS (Signal Distribution System section) and the Design Groundrules, perform the MU Signal Distribution Board (A) detail logic and circuit designs; prepare schematics/logic diagrams, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Assist in the preparation of the detail MU Signal Distribution System section for the Part II MU DS prepared under Task 2.2.3.					
3.3.6	<u>MU Signal Distribution Board (B)</u> In accordance with the MU DS (Signal Distribution System section) and the Design Groundrules, perform the MU Signal Distribution Board (B) detail logic and circuit designs; prepare schematics/logic diagrams, generate intra and inter board interconnect wire lists, provide drafting coordination, review drafting generated documentation, prepare detail test procedures, prepare brief discourses on theory of operation, and support the software development activity in defining diagnostic requirements. Assist in the preparation of the detail MU Signal Distribution System section for the Part II MU DS prepared under Task 2.2.3.					

FORM NO. 16, PT-004 11/20/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE 07	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER 8	
DATE August, 1976									
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
3.4	POWER SYSTEM								
3.4.1	Power Supply Selection								
	Analyze and estimate the voltage and power requirements for the CU and MU. From the analysis, prepare individual detailed Procurement Specifications for each of the power supply types. The Procurement Specification format shall, in general, follow the format defined for Part I DS's per Aero Design Procedure 2.1. The Procurement Specification shall emphasize low cost, off-the-shelf designs, high efficiency, stability, and reliability and ease of maintenance.								
3.4.2	CU Power Distribution System (PDS)								
	In accordance with the CU DS (Power System Section), the Design Groundrules, and the power supply requirements defined under Task 3.4.1, perform the detail electrical design for the CU cabinet PDS including the power cables (internal and external to the cabinet) and load distribution cables.								
	This task shall include the electrical design effort associated with meeting system EMC design requirements, generation of power cable drawings, coordination with mechanical engineering relating to the mechanical design aspects of the CU power distribution and control, coordination with drafting, review of drafting-generated documentation, specification of connectors, analyses of power loss and voltage drops, preparation of test procedures for the PDS, and preparation of the detail CU Power System section for the Part II CU DS prepared under Task 2.2.2.								

FORM NO. 74-PT-004 11/26/66

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE 07	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER 8	
DATE August, 1976									
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
3.4.3	MU Power Distribution System (PDS)								
	In accordance with the MU DS (Power System section), the Design Groundrules, and the power supply requirements defined under Task 3.4.1, perform the detail electrical design for the MU Cabinet PDS including the power cables (internal and external to the cabinet) and load distribution cables/bus bars.								
	This task shall include the electrical design effort associated with meeting system EMC requirements, generation of power cable drawings, coordination with mechanical engineering relating to the mechanical design aspects of the MU power distribution and control, coordination with drafting, review of drafting-generated documentation, specification of connectors, analysis of power loss, voltage drop, and heating effects of the bus bars, preparation of test procedures for the PDS, and preparation of the detail MU Power System section for the Part II MU DS prepared under Task 2.2.3.								

FORM NO. 74-PT-004 11/26/66

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.D. NUMBER		DATE		PAGE OF	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER	
						B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
4.0	<u>MECHANICAL DESIGN</u>								
4.1	<u>CONTROL UNIT</u>								
4.1.1	<u>Structural Design</u> In accordance with the CU DS and the Design Groundrules, perform the CU cabinet structural design. The detail structural design shall be based on the layout concepts defined under Task 4.1.10.								
4.1.2	<u>EMC Design</u> In accordance with the CU DS and the Design Groundrules, generate a mechanical design approach that is compatible with the system EMC requirements. This will include the following areas: <ul style="list-style-type: none"> Input power filtering Cable harness routing Wire types Grounding scheme Chassis bonding to ground Shield terminations Enclosure design 								
4.1.3	<u>Connectors</u> Define the mechanical requirements for the internal and external CU connectors. Define the optimum location/grouping for the various connectors.								

FORM NO. 74-91-004 11/20/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.D. NUMBER		DATE		PAGE OF	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER	
						B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
4.1.4	<u>Power System</u> Evaluate methods of routing power throughout the CU cabinet assembly. This task will address the following: <ul style="list-style-type: none"> Input filtering requirements Method of routing throughout assembly Bus bars Stranded wire Size Material In addition, the power supply mounting provisions, accessibility, and other mechanical/thermal aspects are to be taken into account. From the above, a firm mechanical design for the CU power system shall be performed.								
4.1.5	<u>Signal Distribution</u> Based on the Task 4.1.10 layout concepts associated with the signal distribution aspects (backplanes, card cages, wire/cable type, routing and clamping), perform the detail mechanical design of same.								
4.1.6	<u>Logic Board Design</u> This task pertains to all CU logic/memory circuit boards. The functions shall include: <ul style="list-style-type: none"> Host interface CU signal distribution Slave controller (including program memory) 								

FORM NO. 74-91-004 11/20/62

TASK PLAN CONFIGURATION SHEET		PROGRAM		S.O.W. NUMBER	DATE	PAGE OF	
PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	DATE	
					B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
4.1.6	<u>Logic Board Design (Continued)</u> Based on the layout concepts and circuit board trade-off studies performed under Task 4.1.10, design/define the general-purpose CU circuit board. Define the groundrules for design, layout and method of interconnect for same. The method of connecting the boards to the backplane shall be coordinated with the effort in Task 4.1.5.						
4.1.7	<u>Control Panel</u> Based on the layout concepts performed under Task 4.1.10, perform a detail mechanical design of the control panel.						
4.1.8	<u>Thermal Design/Analysis</u> Define the power requirements for each subassembly comprising the CU cabinet. (This also applies to the peripherals.) Based on the power requirements, develop air cooling concepts for the CU cabinet and peripherals to include volume of air required as well as air distribution concepts. Perform a thermal analysis of the CU and its major components to insure that adequate cooling is provided. From the thermal analysis, finalize the CU thermal design approach. In addition, define the pressure drop requirements.						

FORM NO. 70-PT-004 11/20/68

TASK PLAN CONFIGURATION SHEET		PROGRAM		S.O.W. NUMBER	DATE	PAGE OF	
PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	DATE	
					B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
4.1.9	<u>CU Documentation</u> For each of the individual board assembly and detail designs referenced under Task 4.1.6, prepare and check layouts, assembly drawings, detail drawings, artwork and schematics to document the CU circuit board designs. For the remaining CU cabinet items, prepare concept layouts to document the design approach. From the concept layouts, prepare complete detail and assembly documentation. All final documentation shall be EO-released.						
4.1.10	<u>CU Detail Layout Concept</u> Perform detail layout concepts for all facets of the CU. Concepts shall be formulated for partitioning the CU cabinet assemblies in a manner that will afford the most natural combination of system functions. Concepts shall also be formulated for mounting and providing the necessary mechanical interfaces for the major subassemblies and functions while maintaining the EMC integrity and maintainability provisions. The subassemblies and functions include: <ul style="list-style-type: none"> . CU power system distribution and components . Circuit boards . Signal distribution system and components (including backplanes) . Card cages . Control panel . Minicomputer, main memory, peripherals The concepts for implementing the backplane and card cage assemblies shall be established. Consideration shall be given to using existing designs.						

FORM NO. 70-PT-004 11/20/68

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.W. NUMBER	DATE	PAGE	OF
		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER
					B	August, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER
4.1.10	<p><u>CU Detail Layout Concept</u> (Continued)</p> <p>Also under this task, the methods of routing signals throughout the CU shall be evaluated. The following shall be addressed:</p> <ul style="list-style-type: none"> Stranded wire usage Shielded wire usage Tape cable usage Method of routing and clamping Location of external interface connectors <p>Perform a design trade-off study to determine the best method of packaging the circuit board functions. Consideration shall be given to utilizing existing general-purpose board designs (Augat, etc.) as well as Aero-designed general-purpose boards. From the trade-offs, select the optimum approach.</p> <p>This task also includes the generation of a mechanical layout concept for the Control Panel.</p>					

FORM NO. 74-PT-004 11/26/62

TASK PLAN CONTINUATION SHEET		PROGRAM	S.O.W. NUMBER	DATE	PAGE	OF
		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER
					B	August, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER
4.2	MEMORY UNIT					
4.2.1	<p><u>Structural Design</u></p> <p>In accordance with the MU DS and the Design Groundrules, perform the MU cabinet structural design. The detail design shall be based on the layout concepts defined under Task 4.2.11.</p>					
4.2.2	<p><u>EMC Design</u></p> <p>In accordance with the MU DS and the Design Groundrules, generate a mechanical design approach that is compatible with the system EMC requirements.</p> <p>This will include the following areas:</p> <ul style="list-style-type: none"> Input power filtering Cable harness routing Wire types Grounding scheme Chassis bonding to ground Shield terminations Enclosure design 					
4.2.3	<p><u>Connectors</u></p> <p>Define the mechanical requirements for the internal and external MU connectors. Define the optimum location/grouping for the various connectors.</p>					

FORM NO. 74-PT-004 11/26/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER	
4.2.4	<u>Power System</u> Evaluate methods of routing power throughout the MU cabinet assembly. This task will address the following: . Input filtering requirements . Method of routing throughout assembly . Bus bars . Stranded wire . Size . Material In addition, the power supply mounting provisions, accessibility and other mechanical thermal aspects are to be taken into account. From the above, a firm mechanical design for the MU power system shall be performed.								
4.2.5	<u>Signal Distribution</u> Based on the Task 4.2.11 layout concepts associated with the signal distribution aspects (backplanes, card cages, wire/cable type, routing and clamping), perform the detail mechanical design of same.								
4.2.6	<u>Hybrid Coordination</u> Perform coordination and liaison with the hybrid development activity under Task 3.2.5 to insure that characteristics of the hybrid device are known and understood. Cognizance of all hybrid thermal and general design problems shall be part of this task. The process methods shall be understood. Insure that the hybrid package is compatible with the memory board packaging approach. Make recommendations as required to insure the hybrid/board compatibility.								

FORM NO. 74-71-004 11/18/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER	
4.2.7	<u>Memory and Logic Board Design</u> Based on the conceptual designs and circuit board trade-off studies performed under Tasks 4.2.10 and 4.2.11, perform the detail mechanical designs for the Memory Board and general-purpose logic boards. Define the groundrules for design, layout and method of interconnect for same. The method of connecting the boards to the backplane shall be coordinated with the effort in Task 4.2.5.								
4.2.8	<u>Thermal Design Analysis</u> Define the power requirements for each sub-assembly comprising the MU cabinet. Based on the power requirements, develop air cooling concepts for the MU cabinet. Define the volume of air required as well as air distribution concepts. Perform a thermal analysis of the MU and its major components to insure that adequate cooling is provided. From the thermal analysis, finalize the MU thermal design approach. In addition, define the pressure drop requirements.								
4.2.9	<u>MU Documentation</u> For each of the individual board assembly and detail designs referenced under Task 4.2.7, prepare and check layouts, assembly drawings, detail drawings, artwork and schematics to document the MU circuit board designs. For the remaining MU cabinet items, prepare concept layouts to document the design approach. From the concept layouts, prepare complete detail and assembly documentation. All final documentation shall be EO released.								

FORM NO. 74-71-004 11/18/62

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER	
								B AUGUST, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
4.2.10	<p><u>Memory Board/Hybrid Packaging Concepts</u></p> <p>Perform detail layout concepts for packaging the hybrid and memory board functions. Establish the concepts for the method of heat conduction from the hybrid to the memory board. Establish the concepts for mounting the hybrids to the memory board. Also, establish concepts for the routing of power and ground on the memory board.</p>								
4.2.11	<p><u>MU Detail Layout Concept</u></p> <p>Perform detail layout concepts for all facets of the MU. Concepts shall be formulated for partitioning the MU cabinet assemblies in a manner that will afford the most natural combination of system functions. Concepts shall also be formulated for mounting and providing the necessary mechanical interfaces for the major subassemblies and functions while maintaining the EMC integrity and maintainability provisions. The subassemblies and functions include:</p> <ul style="list-style-type: none"> . MU power system distribution and components . Circuit boards . Signal distribution system and components (including backplanes) . Card cages <p>The concepts for implementing the backplane and card cage assemblies shall be established. Consideration shall be given to using existing designs.</p>								

FORM NO. 14-1-60 11/28/61

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE	
PREPARED BY		DATE		APPROVED BY		DATE		REVISION NUMBER	
								B AUGUST, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
4.2.11	<p><u>MU Detail Layout Concept (Continued)</u></p> <p>Also under this task, the methods of routing signals throughout the MU shall be evaluated. The following shall be addressed:</p> <ul style="list-style-type: none"> . Stranded wire usage . Shielded wire usage . Tape cable usage . Method of routing and clamping . Location of external interface connectors <p>Perform a design trade-off study to determine the best method of packaging the MU signal distribution circuit board functions. Consideration shall be given to utilizing existing general-purpose board designs (Augat, etc.) as well as Aero-designed general-purpose boards. From the trade-offs, select the optimum approach.</p>								

FORM NO. 14-1-60 11/28/61

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	S. O. W. NUMBER	PROJECT AUTHORITY NUMBER	DATE
							B		August, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S. O. W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
4.3	EXTERNAL CABLES								
4.3.1	<u>Cable Design</u> Develop the conceptual approach for all inter-cabinet cables. This includes the cabling required between: <ul style="list-style-type: none"> . 6080 Host Computer and CU cabinet . CU cabinet and MU cabinet(s) It is assumed that an existing cable set will be used to interconnect the peripherals and minicomputer. Under this task, the concepts for the external power cables shall also be developed. As a minimum, the following requirements shall be established: <ul style="list-style-type: none"> . Wire size . Shielding . Jackets . Routing The requirements delineated in the overall system DS and the Design Groundrules shall be complied with to insure that the basic requirements including EMC are met. After the concepts are firmly established, the external cable mechanical design shall be finalized.								
4.3.2	<u>Cable Documentation</u> Prepare complete detail and assembly documentation for the cable designs defined under Task 4.3.1. All cable documentation shall be EO-released.								

FORM NO. 74-01-004 11/10/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	S. O. W. NUMBER	PROJECT AUTHORITY NUMBER	DATE
							B		August, 1976
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S. O. W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
4.4	INTERFACE								
4.4.1	<u>Mechanical Interface</u> Generate a mechanical layout to depict the mechanical interface of the CU, MU and peripherals within the ECAM system installation area. All special mounting provisions and restraints shall be identified. The interface information shall be incorporated in the overall CU and MU structure designs.								
4.4.2	<u>Cooling Interface</u> Define the cooling air inlet and outlet interfaces. Special requirements (such as ducting) for the outlet air shall be defined and incorporated into the two cabinet designs. Mobility of the cabinets for maintenance operations shall be considered if ducting is required.								
4.4.3	<u>Electrical Interface</u> Under this task, the method of providing the electrical interfaces between the ECAM equipments shall be defined. For example, it shall be ascertained if cabling under the floor, on the floor, or overhead is to be provided and the restraint the required approach has on the mobility of the equipments (e.g. during maintenance and troubleshooting). The electrical interface definition shall be used in defining the external connector locations on the CU and MU cabinets.								

FORM NO. 74-01-004 11/10/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
							B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
5.0	<u>SYSTEM SOFTWARE AND ANALYSIS</u> NOTE: Task 5.0 will be a major subcontracted package of work. Detail descriptions of subtasks 5.1 through 5.8 will be defined at a later time and will be in the form of a Statement of Work to the software house subcontractor.								

FORM NO. 74-PT-004 11/28/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
							B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
6.0	<u>SUPPORT SOFTWARE</u>								
6.1	MPC DVT SOFTWARE								
6.1.1	<u>Host Interface/Simulator</u> In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS 00XXXXX1) for the DVT software for the CU Host Interface/Simulator function. Based on the DVT DS, design the DVT software. The appropriate hardware designers shall be consulted as part of the design effort. Code the DVT in MCF assembly language. Debug the DVT using the MCF computer. Document the DVT in the form of listings, flow charts, and a card deck. The listings and flow charts shall be included in Part II of the DVT DS.								
6.1.2	<u>Host Interface/6080</u> In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS 00XXXXX1) for the DVT software for the CU Host Interface/6080 function. Based on the DVT DS, design the DVT software. The appropriate hardware designers shall be consulted as part of the design effort. Code the DVT in MCF assembly language. Debug the DVT using the MCF computer. Document the DVT in the form of listings, flow charts, and a card deck. The listings and flow charts shall be included in Part II of the DVT DS.								

FORM NO. 74-PT-004 11/28/62

TASK PLAN CONTINUATION SHEET		PROJECT NAME		S. O. W. NUMBER		DATE		PAGE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S. O. W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
6.3.1	DVT Software Interrelation (Continued)								
	the linkage and control software in MCP/GMAP assembly language. Debug the software using the MCP and 6080 computers. Document the System Confidence Test in the form of listings, flow charts, operator's instructions, card deck, and a mag tape. The listings and flow charts shall be included in Part II of the System Confidence Test DS.								
6.4	MCP DIAGNOSTIC SOFTWARE								
6.4.1	<u>Host Interface/6080</u>								
	In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the CU Host Interface/6080 function. Based on the DS, design the diagnostic software. The diagnostic software design shall rely heavily on the design of the corresponding DVT software developed under Task 6.1.2 and shall be capable of isolating failures to the board level. Code the diagnostic software in MCP assembly language. Debug the software using the MCP computer. Document the diagnostic software in the form of listings, flow charts, and a card deck. The listings and flow charts shall be included in Part II of the above DS.								

FORM NO. 78-11-001 11/28/62

TASK PLAN CONTINUATION SHEET		PROJECT NAME		S. O. W. NUMBER		DATE		PAGE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S. O. W. PARA. NUMBER	PROJECT AUTHORITY NUMBER	
6.4.2	<u>Slave Controller</u>								
	In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the CU Slave Controller function. Based on the DS, design the diagnostic software. The diagnostic software design shall rely heavily on the design of the corresponding DVT software developed under Task 6.1.4 and shall be capable of isolating failures to the board level. Code the diagnostic software in MCP assembly language. Debug the software using the MCP computer. Document the diagnostic software in the form of listings, flow charts and a card deck. The listings and flow charts shall be included in Part II of the above DS.								
6.4.3	<u>CU Signal Distribution</u>								
	In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the CU Signal Distribution function. Based on the DS design the software. Coordinate the design effort with the appropriate hardware designers. The diagnostic software shall be capable of isolating failures to the board level. Code the diagnostic software in MCP assembly language. Debug the software using the MCP computer. Document the diagnostic software in the form of listings, flow charts and a card deck. The listings and flow charts shall be included in Part II of the above DS.								

FORM NO. 78-11-001 11/28/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
							B	August, 197	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
6.4.4	<u>MU Signal Distribution (A)</u> In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the MU Signal Distribution (A) function. Based on the DS, design the diagnostic software. Coordinate the design effort with the appropriate hardware designers. The diagnostic software shall be capable of isolating failures to the board level. Code the diagnostic software in MCP assembly language. Debug the software using the MCP computer. Document the diagnostic software in the form of listings, flow charts, and a card deck. The listings and flow charts shall be included in Part II of the above DS.								

FORM NO. 70-PT-100 11-60/60

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
							B	August, 197	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
6.4.5	<u>MU Signal Distribution (B)</u> In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the MU Signal Distribution (B) function. Based on the DS, design the diagnostic software. Coordinate the design effort with the appropriate hardware designers. The diagnostic software shall be capable of isolating failures to the board level. Code the diagnostic software in MCP assembly language. Debug the software using the MCP computer. Document the diagnostic software in the form of listings, flow charts, and a card deck. The listings and flow charts shall be included in Part II of the above DS.								
6.4.6	<u>Array</u> In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the MU memory board Array function. Based on the DS, design the diagnostic software. The diagnostic software design shall rely heavily on the design of the corresponding DVT software developed under Task 6.1.4 and shall be capable of isolating failures to the board level. Code the diagnostic software in MCP assembly language. Debug the software using the MCP computer. Document the diagnostic software in the form of listings, flow charts, and a card deck. The listings and flow charts shall be included in Part II of the above DS.								

FORM NO. 70-PT-100 11-60/60

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	PROJECT NUMBER	PAGE	OF
							B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
6.5	<u>6080 DIAGNOSTIC SOFTWARE</u>								
6.5.1	<u>Host Interface/MCP</u> In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the diagnostic software for the CU Host Interface/MCP function. Based on the DS, design the diagnostic software. The diagnostic software design shall rely heavily on the design of the corresponding DVT software developed under Task 6.2.1 and shall be capable of isolating failures to the board level. Code the diagnostic software in GMAP assembly language. Debug the software using the 6080 computer. Document the diagnostic software in the form of listings, flow charts and a card deck. The listings and flow charts shall be included in Part II of the above DS.								

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	PROJECT NUMBER	PAGE	OF
							B	August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
6.6	<u>SYSTEM MAINTENANCE SOFTWARE</u>								
6.6.1	<u>Diagnostic Software Integration</u> Under this Task, the diagnostic software developed under Tasks 6.4 and 6.5 shall be integrated into a single software package described as the System Maintenance Software. It shall serve as a maintenance tool in diagnosing and isolating subsystem failures. In accordance with Aero Division Design Procedure 2.1, prepare a Part I and II Computer Program Detail Specification (DS QGXXXXA1) for the System Maintenance Software. Design the necessary linkages and controls to execute the diagnostic software modules as a total entity or individually. Code the linkage and control software in MCP/GMAP assembly language. Debug the software using the MCP and 6080 computers. Document the System Maintenance Software in the form of listings, flow charts, operator's instructions, card deck, and a mag tape. The listings and flow charts shall be included in Part II of the above DS. This task includes the final debug and rendering the system maintenance software operational while the equipment is at the interim 6080 site.								
6.7	<u>AERO SDF MINICOMPUTER AND PERIPHERAL PROCUREMENT</u> Procure the minicomputer, peripheral complement and other items associated with the Aero SDF.								

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVAL BY		DATE		S.O.W. NUMBER		PAGE		OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARA. NUMBER		PROJECT AUTHORITY NUMBER			
8.0		<u>TEST</u>													
8.1		<u>COMPONENT TEST</u>													
8.1.1		<u>Memory Device Characterization</u> Perform characterization tests of the selected memory device using 10 sample devices. The following three categories of tests and subtests shall be performed: 1. <u>Normal Operating Characteristics Tests</u> . Voltage variation . Clock timing variation . Interface voltage/current level variation . Pattern sensitivity . Frequency range of operation . Optimum operating point . Temperature range . Control timing variation . Refresh period range 2. <u>Destruction Tests</u> . Absolute maximum voltage . Maximum temperature 3. <u>Application Tests</u> . Noise sensitivity . Parallel bus operation The design and fabrication of a device interface to adapt the memory device to the device tester and programming of the device shall be included under Tasks 11.1.5 and 11.2.5. This task shall include data analysis, curve plotting, and preparation of a summary report.													

FORM NO. 16-PT-004 11/10/68

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE		APPROVAL BY		DATE		S.O.W. NUMBER		PAGE		OF	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.		SCHEDULE		PROGRAM ELEMENT NUMBER		S.O.W. PARA. NUMBER		PROJECT AUTHORITY NUMBER			
8.1.2		<u>WLIC Characterization</u> Perform characterization tests of the Word Logic LSIC device using 10 sample devices which are to be furnished under Task 3.2.1.10. The following three categories of tests and subtests shall be performed: 1. <u>Normal Operating Characteristics Tests</u> . Voltage variation . Clock timing variation . Interface voltage/current level variation . Frequency range of operation . Optimum operating point . Temperature range . Control timing variation . I/O transfer function 2. <u>Destruction Tests</u> . Absolute maximum voltage . Maximum temperature 3. <u>Application Tests</u> . Noise sensitivity . Parallel bus operation The design and fabrication of a device interface to adapt the word logic LSIC to the device tester and programming of the device shall be included under Tasks 11.1.4 and 11.2.4. This task shall include data analysis, curve plotting, and preparation of a summary report.													

FORM NO. 16-PT-004 11/10/68

TASK PLAN CONTINUATION SHEET		DATE		DATE		DATE		DATE		DATE		DATE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER					
8.1.3	<u>Hybrid Characterization</u> Perform characterization tests of the memory Hybrid device using five sample hybrid devices which are to be furnished under Task 3.2.3.13. The following three categories of tests and subtests shall be performed: 1. <u>Normal Operating Characteristics Tests</u> . Voltage variation . Clock timing variation . Interface voltage/current level variation . Frequency range of operation . Optimum operating point . Temperature range . Control timing variation . I/O transfer function 2. <u>Destruction Tests</u> . Absolute maximum voltage . Maximum temperature 3. <u>Application Tests</u> . Noise sensitivity . Parallel bus operation The design and fabrication of a device interface to adapt the hybrid to the device tester and programming of the device tester shall be included under tasks 11.1.2 and 11.2.2. This task shall also include data analysis, curve plotting, and preparation of a summary report.												

FORM NO. 14-1-100-1 11/20/62

TASK PLAN CONTINUATION SHEET		DATE		DATE		DATE		DATE		DATE		DATE	
TASK NO. AND NAME		DESCRIPTION OF EFFORT		DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER					
8.1.4	<u>Hybrid Test</u> Following fabrication and prior to installation of the Hybrid device on the Memory Board, each Hybrid device shall pass a functional test in accordance with the test procedure prepared under Task 3.2.3.2. The high speed static General Radio board tester (or equivalent) shall be used to perform the room ambient tests. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.2 and 11.2.2.												
8.2	<u>SUBUNIT TEST</u>												
8.2.1	<u>Slave Controller</u> Using the General Radio board tester as test equipment, perform a high speed static functional test on the Slave Controller in accordance with the test procedures prepared under Task 3.1.3. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.1 and 11.2.1.												
8.2.2	<u>Host Interface</u> Using the General Radio board tester as test equipment, perform a high speed static functional test on the Host Interface in accordance with the test procedures prepared under Task 3.1.1. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.1 and 11.2.1.												

FORM NO. 14-1-100-1 11/20/62

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
						B	August, 197	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
8.2.3	<u>MCP Computer and Peripherals</u> Connect the MCP computer to the peripherals (keyboard/display, line printer, and mag tape unit) using the standard cables provided with the equipment. Perform the standard "confidence/diagnostic" software tests provided with the equipment to insure that the computer, peripherals, and cabling are all operational prior to their installation as part of the CU.							
8.2.4	<u>CU Signal Distribution Board</u> Using the General Radio board tester as test equipment, perform a high speed static functional test on the CU Signal Distribution Board in accordance with the test procedures prepared under Task 3.3.4. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.1 and 11.2.1.							
8.2.5	<u>Control Panel</u> Using manual bench equipment (meters, scopes, etc.), checkout and debug the Control Panel in accordance with the test procedures prepared under Task 3.1.5.							
8.2.6	<u>Memory Board</u> Using the General Radio board tester as test equipment, perform a high speed static functional test on the Memory Board in accordance with the test procedures prepared under Task 3.2.4. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.1 and 11.2.1.							

FORM NO. 70-PT-004 11/10/68

TASK PLAN CONTINUATION SHEET		PREPARED BY	DATE	APPROVED BY	DATE	REVISION NUMBER	PAGE	OF
						B	August, 197	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER		
8.2.7	<u>MU Signal Distribution Board (A)</u> Using the General Radio board tester as test equipment, perform a high speed static functional test on the MU Signal Distribution Board (A) in accordance with the test procedures prepared under Task 3.3.5. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.1 and 11.2.1.							
8.2.8	<u>MU Signal Distribution Board (B)</u> Using the General Radio board tester as test equipment, perform a high speed static functional test on the MU Signal Distribution Board (B) in accordance with the test procedures prepared under Task 3.3.6. Provision of a device adapter and programming of the board tester shall be provided for under Tasks 11.1.1 and 11.2.1.							
8.2.9	<u>CU Cabinet Power System</u> After cabinet installation of the power supplies, filters, power cables/bus bars, and other cabling/equipment associated with the CU Power System, perform a Power System electrical test to insure that there are no "opens", "shorts" or misconnections and that all voltages are of the correct value and properly distributed.							

FORM NO. 70-PT-004 11/10/68

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE 01	
PREPARED BY		DATE		APPROVED BY		DATE		REV. S.O.W. NUMBER	
						B		August, 1974	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
8.2.10	<u>CU Cabinet Signal Distribution</u> After cabinet installation of the signal distribution wiring and cabling, perform a continuity check to insure that the wire routing is, in general, correct and that there are no inadvertent connections to the cabinet power system. Final correctness of the signal distribution wiring will be verified during the "unit level" tests.								
8.2.11	<u>MU Cabinet Power System</u> After cabinet installation of the power supplies, filters, power cables/bus bars, and other cabling/equipment associated with the MU Power System, perform a Power System electrical test to insure that there are no "opens", "shorts", or misconnections and that all voltages are of the correct value and properly distributed.								
8.2.12	<u>MU Cabinet Signal Distribution</u> After cabinet installation of the signal distribution wiring and cabling, perform a continuity check to insure that the wire routing is, in general, correct and that there are no inadvertent connections to the cabinet power system. Final correctness of the signal distribution wiring will be verified during the "unit level" tests.								
8.2.13	<u>Cable Set</u> Perform a continuity test on the cable set wiring to insure that there are no "opens", "shorts", or misconnections.								

FORM NO. PG-11-004 11/20/63

TASK PLAN CONTINUATION SHEET		PROGRAM		S.O.W. NUMBER		DATE		PAGE 01	
PREPARED BY		DATE		APPROVED BY		DATE		REV. S.O.W. NUMBER	
						B		August, 1974	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER			
8.3	<u>UNIT TEST</u>								
8.3.1	<u>MCP/Host Interface/6080 Simulator</u> In accordance with the test plans and procedures prepared under Tasks 2.6.2 and 2.6.3 respectively, the complete CU Host Interface circuitry shall be verified by exercising the circuitry via the 6080 Simulator and the MCP. The Host Interface/Simulator DVT software provided under Task 6.1.1 shall be used for this purpose.								
8.3.2	<u>MCP and Slave Controller</u> In accordance with the test plans and procedures prepared under Tasks 2.6.2 and 2.6.3 respectively, verify the functional operation of the combined MCP and Slave Controller circuitry. The Slave Controller DVT software provided under Task 6.1.3 shall be used for this purpose.								
8.3.3	<u>CU and MU</u> After the CU functional operation has been verified under Tasks 8.3.1 and 8.3.2, perform a functional checkout of the connected CU and MU cabinets. The Array DVT software provided under Task 6.1.4 shall be used for this purpose. The tests shall be conducted in accordance with the test plans and procedures prepared under Tasks 2.6.2 and 2.6.3 respectively.								

FORM NO. PG-11-004 11/20/63

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	S.O.W. NUMBER		DATE	OF
							B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER				
8.4	SYSTEM TEST									
8.4.1	<u>Transport to Interim Site</u> Transport the CU, MU and associated equipment to an interim 6080 site for system level integration tests. This task shall include the packaging, transportation, and unpackaging of the equipment at the interim site.									
8.4.2	<u>System Integration and C/O W/Support Software</u> In accordance with the test plans and procedures prepared under Tasks 2.6.2 and 2.6.3 respectively, set-up, integrate and test the complete deliverable system hardware at the interim 6080 site. For this purpose, the System Confidence Test (SCT) software developed under Task 6.3 shall be used. (NOTE: The SCT software shall be fully debugged and verified under this task.) Also under this task, the diagnostic software developed under Tasks 6.4 and 6.5 shall be debugged and verified.									
8.4.3	<u>System C/O W/System Software</u> This task is essentially a support effort to the System Software debug effort under Tasks 5.6.1, 5.6.2 and 5.6.3. Under this task, all final hardware/software design changes are to be accomplished.									

FORM NO. 78-PT-006 11/26/62

TASK PLAN CONTINUATION SHEET		PREPARED BY		DATE	APPROVED BY	DATE	S.O.W. NUMBER		DATE	OF
							B		August, 1976	
TASK NO. AND NAME	DESCRIPTION OF EFFORT	DEPT. OR GROUP RESP.	SCHEDULE	PROGRAM ELEMENT NUMBER	S.O.W. PARA. NUMBER	PROJECT AUTHORITY NUMBER				
8.4.4	<u>In-House Acceptance Test</u> In accordance with the test plans and procedures prepared under Tasks 2.6.2 and 2.6.3 respectively, perform a final "in-house" acceptance test at the interim 6080 site. For this purpose, the System Confidence Test software prepared under Task 6.3 shall be used.									

FORM NO. 78-PT-006 11/26/62

7.4 SYSTEM SOFTWARE SUBCONTRACTOR SOW

7.4.1 Scope

This effort has two major objectives. The first is to analyze ECAM performance in detail sufficient to allow design improvements, and the second is to develop the system software necessary to operate the ECAM in the application environment. This work makes up Task 5 of the Contractor's Program Plan.

7.4.2 Statement of Work

7.4.2.1 Preliminary System Software Spec (Honeywell Phase 1, Subtask 5.1)--

The Subcontractor shall prepare a preliminary specification of the ECAM-resident software and the required support tools. This shall include, but is not limited to, descriptions of the software for the following functions:

- Master Processor:
 - Host Computer Interface
 - Query Processing:
 - Mapping to slave language
 - Error handling
 - Master Memory Management:
 - Host communication buffers
 - Slave communication buffers
 - Slave working storage
 - Master working storage
 - Master Executive Functions:
 - Master processor scheduling and dispatching
 - Slave processor scheduling and dispatching
 - ECAM Initialization:
 - Descriptor table initialization
 - Data base manager interface
 - Multi-User Bookkeeping:
 - State-saving in master
 - Match-stack allocation in ECAM
 - Garbage collection
 - Maintenance and Diagnostic Software Interface

- Host Processor:

- Host I/O Subsystem Modules (e.g., device drivers)
- Fast I/O Software
- User Job Interface Protocols

The level of specification shall be sufficient to allow timing estimates to be made as input to the system simulation subtask (5.2).

In addition, the Subcontractor shall develop a model of the system user characteristics as input to the simulation. These shall include, but are not limited to, the following:

- Normal and worst-case frequency distributions on:
 - Queries
 - Updates
 - Deletions
 - Checkpoint/restart operations
- Query complexity characterization and distributions
- Record sizes and quantities

Draft documentation of the results of Subtask 5.1 is required 3 months after contract award, with final documentation 30 days thereafter.

7.4.2.2 Simulation and Analysis (Honeywell Phase 1, Subtask 5.2) --
Using the data developed in Subtask 5.1, together with hardware performance data provided by the Prime Contractor, the Subcontractor shall conduct a functional simulation of the ECAM operating in the target environment. The purpose of this simulation is identification of software and hardware problems early enough so that they may be eliminated. The simulation shall be conducted in detail sufficient to develop usage distributions on the following:

- Host I/O subsystem hardware
- ECAM-host interface hardware
- Master-Control Processor
- Interpreter
- Iteration Controller
- ECAM array

During and following the simulations, the Subcontractor shall participate with the Prime Contractor in the design of improvements to remove bottlenecks and to identify areas where system costs may be reduced by changes to the hardware and to the software identified in Subtask 5.1. This latter activity is expected to involve design simplifications of software and hardware elements which prove to be "overdesigned."

The Subcontractor shall participate with the Prime Contractor in documentation and review of the redesign work with the Government to ensure that the overall program objectives are met.

Completion of this subtask shall be scheduled for a time consistent with the requirement that detailed specification of hardware and software (Subtasks 2.2.2, 2.2.3, and 5.3) be completed 11 and 12 months after contract start, respectively.

7.4.2.3 Software Functional Specifications (Honeywell Phase 1, Subtask 5.3) --
The preliminary specifications developed under Subtask 5.1 shall be revised to reflect changes made during the simulation task and issued in final form.

Drafts of these documents are required 11 months after contract award, with final versions 1 month thereafter.

7.4.2.4 Detailed Software Specifications (Honeywell Phase 2, Subtask 5.4) --
Detailed software design specifications shall be prepared for each of the modules identified in Subtask 5.3. Any substantive changes made to the modules during the design process shall be evaluated using the functional simulator of Subtask 5.2.

As a part of the specification, the timing estimates of Subtask 5.1 shall be reviewed and the effect of any substantial variances shall be determined by use of the functional simulator.

Draft specifications shall be completed 16.5 months after contract award, with final versions 1 month thereafter.

7.4.2.5 Software Implementation (Honeywell Phase 2, Subtask 5.5) --
Implementation and unit testing of the modules identified in the design specification shall be performed on a schedule that allows integration (Subtask 5.6) to begin 29 months after contract award.

An Implementation Plan showing the Subcontractor's approach to structuring the code, incorporating requirements changes, unit testing, and software configuration control shall be provided to the Prime Contractor 1 month after beginning this task. A list of intermediate milestones shall be developed and mutually agreed upon by Prime and Subcontractor at this point also.

7.4.2.6 Software Integration and Test -- The modules which make up the ECAM system software shall be integrated and tested according to a Test Plan submitted by the Subcontractor.

The Subcontractor shall deliver a Software Acceptance Test Plan 29 months after contract award and shall complete his testing 4 months thereafter. A system software acceptance test will be conducted by the Prime Contractor approximately 33.5 months after contract award.

7.4.2.7 Software Documentation (Honeywell Phase 2, Subtask 5.7) -- The software developed under Subtask 5.6 shall be documented according to standards to be determined (TBD). Final documentation is to be complete at the time of the acceptance tests.

7.4.3 Furnished Equipment and Facilities

A Software Development Facility (SDF) containing the master minicomputer and TBD peripherals will be made available at the Subcontractor's site during the period of Subtasks 5.5 and 5.6.

TBD hours of computer time on the host or on an equivalent machine will also be made available by the Prime Contractor or will be GFE during Subtask 5.5.

Subtask 5.6 (Integration) will be performed on-site at the Government's facility or, if security requirements prevent this, the work will be performed at a site which includes a machine identical to the ultimate host in all major respects.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm	...	V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10 ¹²	tera	T
1 000 000 000 = 10 ⁹	giga	G
1 000 000 = 10 ⁶	mega	M
1 000 = 10 ³	kilo	k
100 = 10 ²	hecto*	h
10 = 10 ¹	deka*	da
0.1 = 10 ⁻¹	deci*	d
0.01 = 10 ⁻²	centi*	c
0.001 = 10 ⁻³	milli	m
0.000 001 = 10 ⁻⁶	micro	μ
0.000 000 001 = 10 ⁻⁹	nano	n
0.000 000 000 001 = 10 ⁻¹²	pico	p
0.000 000 000 000 001 = 10 ⁻¹⁵	femto	f
0.000 000 000 000 000 001 = 10 ⁻¹⁸	atto	a

* To be avoided where possible.

*MISSION
of
Rome Air Development Center*

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C³) activities, and in the C³ areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.



*MISSION
of
Rome Air Development Center*

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C³) activities, and in the C³ areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

